Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Srovnání šablonovacích systémů napříč MVC frameworky

Comparsion of templating systems across MVC frameworks

2014 Jakub Orava

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jakub Orava**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Srovnání šablonovacích systémů napříč MVC frameworky
Comparison of Templating Systems across MVC Frameworks

Zásady pro vypracování:

Cílem této bakalářské práce je porovnat šablonovací systémy view vrstvy dnes nejvíce využívaných webových frameworků. Pro toto srovnání by měly být vybrány běžně rozšířené frameworky a jejich šablonovací systémy napříč programovacími jazyky. Například JSP pro jazyk Java, MVC.NET pro C#, Django pro Python a Nette pro PHP.
Srovnání by mělo zohledňovat:
a) rozšířitelnost,
b) bezpečnost,
c) práci s daty,
d) lokalizaci,
f) testovatelnost.
U každého bodu porovnejte jednotlivé šablonovací systémy, vypište jejich hlavní nedostatky, přednosti a návrhy na vylepšení. Použití jednotlivých šablonovacích systémů ukažte na jednoduchých projektech.

Seznam doporučené odborné literatury:

[1] HALL, Marty. *Java: servlety a stránky JSP*. Praha: Neocortex, 2001, xviii, 585 s. ISBN 80-863-3006-0
[2] FREEMAN, Adam. *Pro ASP.NET MVC 3 framework*. 3rd. ed. New York: Apress, c2011, xxv, 824 s. The expert's Voice in.NET. ISBN 978-1-4302-3404-3.
Nette Framework: dokumentace [online]. 2013. Dostupné z: http://doc.nette.org/cs/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Daniel Robenek**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 2.5.2014 .....

Rád bych na tomto místě poděkoval vedoucímu práce, Ing. Danielu Robenkovi, za všechnu pomoc, ochotu a cenné rady při vypracování této bakalářské práce.

Abstrakt

Tato bakalářská práce se zabývá srovnáním šablonovacích systémů čtyř aplikačních rámců (frameworků) pro vývoj internetových aplikací ve čtyřech různých programovacích jazycích: Nette (PHP), Django (Python), ASP.NET MVC (C#), Ruby on Rails (Ruby). V první části bude čtenář seznámen se základní filozofií frameworků a popisem programovacích jazyků. V části druhé se práce zabývá již samotným srovnáním. Šablonovací systémy vybraných frameworků budou srovnány z pohledu bezpečnosti, rozšířitelnosti, testovatelnosti, tvorby lokalizací a práce s daty. V druhé části jsou nejprve popsány kritéria pro hodnocení, následuje samotné hodnocení a na závěr budou výsledky zhodnoceny a porovnány.

Klíčová slova: framework, šablonovací systém, MVC, PHP, Python, Ruby, C#, Nette, Django, Ruby on Rails, ASP.NET MVC, Razor, Latte, ERB

Abstract

This bachelor thesis deals with comparison of templating systems of four application frameworks for developing web applications in four different programming languages: Nette (PHP), Django (Python) ASP.NET MVC (C#), Ruby on Rails (Ruby). In the first part, the basic philosophy of the frameworks and a description of programming languages will be introduced. The second part of this thesis will be focused on comparison of these frameworks. Templating systems of selected frameworks will be compared in terms of security, scalability, testability, making localization and working with data. First of all, the evaluation criteria are described and in the second part they are evaluated. Finally, the results will be evaluated and compared.

Keywords: framework, templating system, MVC, PHP, Python, Ruby, C#, Nette, Django, Ruby on Rails, ASP.NET MVC, Razor, Latte, ERB

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript And XML
API	–	Application Programming Interface
CoC	–	Convention over Configuration
CSS	–	Cascading Style Sheets
DRY	–	Don´t Repeat Yourself
ERB	–	Embedded Ruby
GPL	–	General Public Licence
HHVM	–	HipHop Virtual Machine
HTML	–	HyperText Markup Language
IDE	–	Integrated Development Environment
KISS	–	Keep It Simple Stupid
MTV	–	Model-Template-View
MVC	–	Model-View-Controller
MVP	–	Model-View-Presenter
NewBSD	–	New Berkeley Software Distribution
ORM	–	Object Relational Mapping
PHP	–	PHP: Hypertext Preprocessor
Rails	–	Ruby on Rails
SEO	–	Search Engine Optimization
SQL	–	Structured Query Language
URL	–	Uniform Resource Locator
XML	–	Extensible Markup Language
XSS	–	Cross-Site Scripting

Obsah

Seznam tabulek

Seznam obrázků

Úvod

Tématem této bakalářské práce je srovnání šablonovacích systémů napříč MVC frameworky. MVC frameworky slouží většinou pro vývoj webových aplikací a programátoři si bez nich v dnešní době nedokáží představit napsat sebemenší aplikaci. Frameworky usnaďnují práci a šetří čas programátorovi, který se může věnovat složitějším problémům.

Téma jsem si zvolil z toho důvodu, že se již tři roky zabývám vývojem internetových aplikací pomocí frameworků. Dalším důvodem výběru byla možnost rozšíření stávajících znalostí nebo nabytí úplně nových, neboť jsem mohl zvolil jazyky i frameworky, které mě zajímaly.

Cílem této práce je, mimo jiné, objasnit, co je to framework a návrhový vzor MVC, na kterém většina frameworků, případně jeho mutacích (MVP, MTV), funguje. Hlavním cílem je ale porovnání frameworků Nette, Django, Ruby on Rails a ASP.NET MVC a jejich šablonovacích systémů, tedy nejdůležitější části View vrstvy v návrhovém vzoru MVC. V porovnání se práce zaměřuje především na bezpečnost, testovatelnost, rozšířitelnost, práci s daty a tvorbu lokalizací v těchto frameworcích. Výsledkem porovnání by nemělo být stanovení absolutního vítěze, ale nalezení výhod jednotlivých frameworků a zjištění, který je lepší, v jednotlivých kategoriích.

Tato práce je rozdělena do dvou bloků, teoretického a praktického. V teoretickém bloku je čtenář v první kapitole seznámen s jednotlivými pojmy, jako je framework, či návrhový vzor. Druhá kapitola práce popisuje jednotlivé programovací jazyky, na kterých jsou vybrané frameworky založeny. Kapitola třetí vystihuje jednotlivé frameworky, jejich výhody, nevýhody a dodržované principy. Mimo jiné je v této kapitole čtenář seznámen i s principy, kterých se framework drží (např. DRY nebo KISS). V bloku praktickém jsou v kapitole čtvrté popsány jednotlivé kategorie pro srovnání frameworků. Pátá kapitola se zabývá již samotným srovnáním a je rozdělena do šesti podkapitol, podle porovnávaných kategorií. V poslední, tedy šesté kapitole jsou zhodnoceny celkové dosažené výsledky frameworků a následně rozepsány srovnání výsledků frameworků v jednotlivých kategoriích.

Práce může mít přínos jak pro začínající, tak i mírně pokročilé vývojáře webových řešení, kteří se zatím k používání frameworků nedostali, případně frameworky používají, ale chtěli by se seznámit podrobněji se šablonovacími systémy.

Pro pochopení práce je potřebná alespoň minimální znalost objektově orientovaného programování a jazyka HTML, případně základní znalost syntaxe programovacíh jazyků použitých ve frameworcích.

1 Základní pojmy

Celá má bakalářská práce se týká MVC (model-view-controller) frameworků, a tak bych chtěl pro začátek vysvětlit tyto dva základní pojmy.

1.1 Framework

Začnu tedy vysvětlením pojmu framework. Definice frameworku není ustálena, existuje proto několik definic a já zde některé uvedu:

- „Framework je množina spolupracujících tříd, které tvoří opakovaně použitelný návrh pro určitou skupinu softwaru.“[1]
- „Framework je množina obecných a předpřipravených softwarových stavebních kamenů, které programátor může používat, rozšiřovat nebo upravovat pro specifické řešení. S frameworky vývojář nemusí začínat vždy od nuly pokaždé, když začíná psát aplikaci. Frameworky jsou vyrobeny z kolekce objektů, takže design i kód frameworku může být použitý znova.“[2]

Z definic vyplývá, že framework obsahuje již předpřipravená řešení, která může programátor do jisté míry upravovat. Dokáže programátorovi usnadnit práci, neboť ho zbaví povinností psát pořád dokola kód pro rutinní úkoly, kterými jsou např. přihlašování, tvorba a zpracování formulářů, odesílání e-mailů, apod. Díky tomu se může programátor zaměřit pouze na důležité problémy.

1.2 Návrhový vzor

Frameworky jsou tvořeny za pomoci návrhových vzorů. Některé frameworky uvedené v mé bakalářské práci jsou tvořeny na základě návrhového vzoru MVC nebo některé jeho mutace (MVP, MTV), a proto bych nyní vysvětlit pojem „Návrhový vzor“ a konkrétně návrhový vzor Model-View-Controller a Model-View-Presenter.

Nejprve bych se rád krátce zmínil o pojmu návrhový vzor. Pokusím se opět uvést několik definic, abychom si udělali představu co tento pojem znamená.

„Návrhový vzor popisuje problém, který se pořád dokola vyskytuje v našem prostředí, a poté popisuje jádro řešení tohoto problému, a to takovým způsobem, že můžete použít toto řešení milionkrát, aniž byste to dělali dvakrát stějným způsobem.“ [3]

Existuje mnoho způsobů, jak aplikace naprogramovat, ale programátor si vybere ten, který je ověřený a funguje u mnoha jeho „kolegů“ po celém světě. A právě tento postup se nazývá návrhový vzor.

Podobně jako u frameworků, i použití návrhových vzorů nám ušetří čas, abychom se mohli zaměřit na hlavní problémy aplikace.

1.3 Model-View-Controller

Model-View-Controller (dále jen MVC) je návrhový vzor (někdy nazýván architektonický vzor), který rozděluje aplikaci na tři části – model, view a controller. Řeší problematiku

tzv. „špagetového kódu“, tedy kódu, ve kterém se nachází zároveň přístup k datům z databáze, operace s těmito daty a HTML značky pro uspořádání těchto dat a pěkný vzhled aplikace. Takovýto kód se velice špatně udržuje a někdy se v něm nevyzná ani programátor sám. Díky MVC, který rozděluje funkční a datový model (Model), prezentaci dat (View) a aplikační či řídící logiku (Controller), máme vše krásně rozděleno, kód programovacího jazyka je oddělen od HTML kódu.

1.3.1 Model

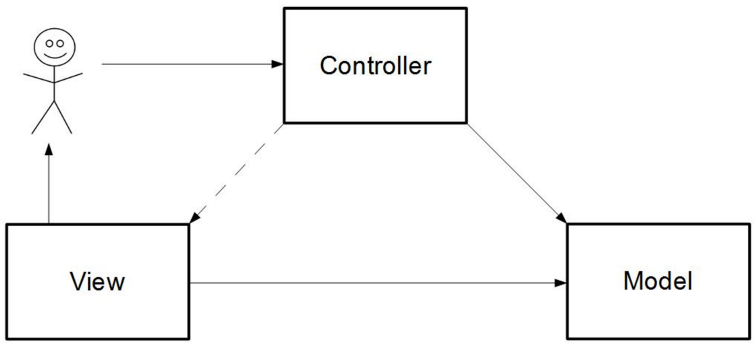
Jak už jsem zmínil výše, modelem se myslí datový model aplikace. Reprezentuje tedy data nebo aplikační logiku, např. databázovou tabulku. V modelu můžeme například definovat pravidla pro validaci přijatých dat z formulářů.

1.3.2 View

View se stará o grafickou stránku aplikace (uživatelské rozhraní). Přistupuje k datům přes Model a říká, jak mají být data prezentována. View si můžeme představit jako okno aplikace, či HTML stránku. View je u webových frameworků většinou reprezentován pomocí šablonovacího systému.

1.3.3 Controller

Controller zajišťuje chování aplikace, zpracovává vstupy a události od uživatele. Jsou zde například definovány funkce pro práci s daty, nebo výpočty.

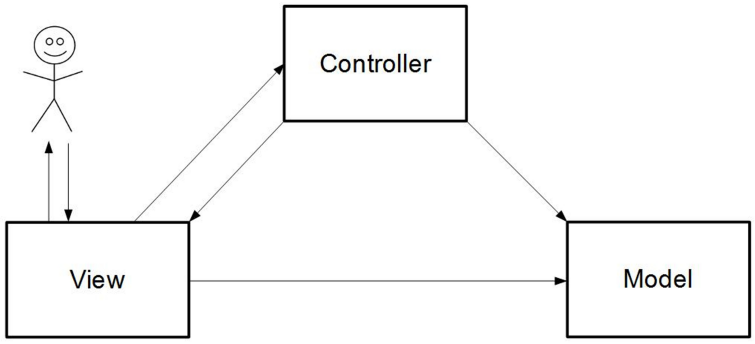


Obrázek 1.1: Návrhový vzor MVC.

1. Uživatel provede akci (vyplnění formuláře, stisknutí tlačítka, . . .)
2. Dotaz zachytí a zpracuje Controller, který zavolá Model pro požadovanou akci, získá a zpracuje data z Modelu.
3. Poté Controller vybere View. Ten pak zobrazí data z Modelu uživateli.

1.4 Model-View-Presenter

Tuto upravenou verzi architektonického vzoru MVC nepoužívá jen framework Nette, ale také Java frameworky jako například Vaadin[4]. Model se oproti MVC nijak nemění. View v MVP vykonává kontrolu uživatelského vstupu a výstupu, oproti View v MVC, které zodpovídalo za tvorbu výstupu. Controller byl nahrazen Presenterem. Presenter vybírá View a předává mu vhodný Model (data). Dále zpracovává reakce uživatelů (změna pohledu nebo stavu).



Obrázek 1.2: Návrhový vzor MVP.

1.5 Další použité návrhové vzory

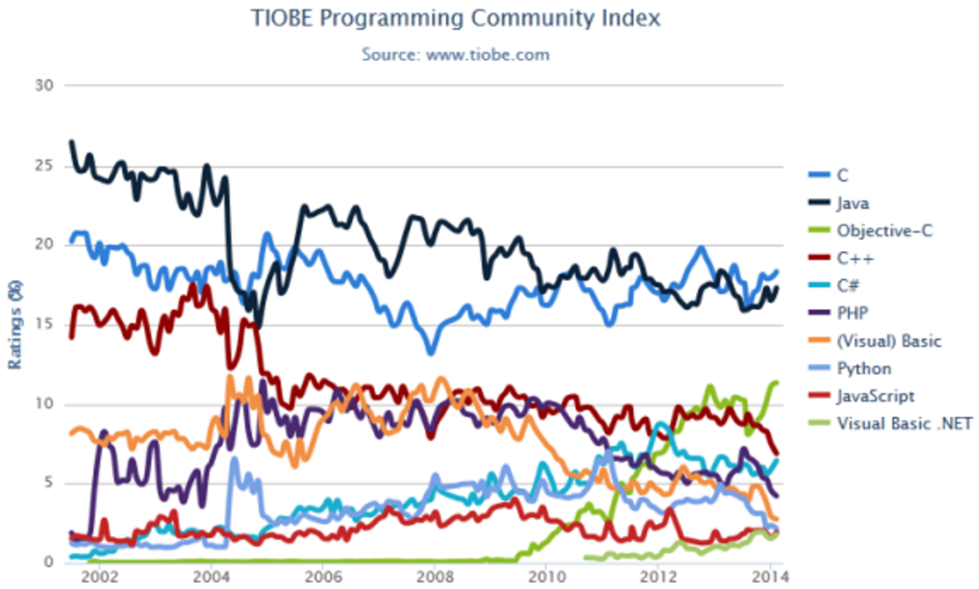
Celý Framework samozřejmě nestojí pouze na návrhovém vzoru MVC nebo MVP. Jsou zde další vzory, které jsou popsány v knize Patterns of Enterprise Application Architecture. Jsou to například vzory Singleton nebo Factory Method zastupující vzory vytvářející, dále návrhový vzor chování Observer nebo také strukturální Composite. Nebudu se o těchto návrhových vzorech více rozepisovat, protože má práce se týká šablonovacích systémů, a ne frameworků kompletně.

2 Použité programovací jazyky

Ve své bakalářské práci jsem si vybral frameworky, které spolupracují s jazyky PHP, Python, Ruby a C#

V této kapitole vás se zmíněnými jazyky seznámím, napíšu něco o jejich historii a současnosti. Dále také uvedu, jaké výhody a nevýhody konkrétní jazyk má.

Nejdříve však odůvodním, proč jsem si pravě tyto jazyky vybral. Jde vidět, že větší zastoupení mají jazyky skriptovací. Důvodem je to, že jejich použití při tvorbě internetových aplikací je zastoupeno více. Na druhou stranu chci uvést alespoň jeden Framework, který je postaven na jazycích kompilovaných, nebo jazycích s virtuálním strojem. PHP jsem zvolil proto, že je nejpoužívanějším skriptovacím jazykem na straně serveru.[5] V jazyce Python jsem programoval ve 3. semestru vysoké školy a Ruby jsem si vybral ze své vlastní zvědavosti a chuti vyzkoušet něco nového. Z jazyků s virtuálním strojem má v mé práci tedy zastoupení jazyk C#. Na škole jsem se učil několika kompilovaným jazykům/jazykům s virtuálním strojem jako C, C++, C# nebo Java. Jazyk C jsem si nevybral, i přesto, že je v dnešní době nejvíce populární, protože pro něj frameworky takřka neexistují. C++ frameworky pro vývoj internetových aplikací má, ale frameworky napsané v jazyce C# nebo Java jsou více propracované a rozšířené. Proto jsem se rozhodoval mezi C# nebo Javou. C# jsem zvolil z důvodu, že jsem v něm dělal všechny školní projekty, kde bylo právě na výběr mezi C# nebo Javou.



Obrázek 2.1: Graf nejpoužívanějších jazyků podle Tiobe [6]

Z obrázku 2.1 lze vidět, že mnou vybrané jazyky se nachází v desítce nejpoužívanějších programovacích jazyků, kromě jazyka Ruby. Ruby jsem si vybral i z toho důvodu, abych porovnal skriptovací jazyk, který se nenachází v první desítce. Je nutno podotknout, že graf byl pořízen v únoru tohoto roku a pozice jazyků se neustále mění.

2.1 PHP

PHP (Hypertext Preprocessor, původně Personal Home Page) je skriptovací jazyk, který nám umožňuje tvořit dynamické internetové stránky. PHP bylo vytvořeno Rasmusem Lerdorfem v roce 1995. Za tu dobu se stal velice populárním a běží na několika milionech domén po celém světě. Nejnovější vydaná verze PHP je 5.5.10 vydána 6. 3. 2014 (dle oficiálních stránek). O všech skriptovacích jazycích se právem tvrdí, že jsou pomalejší, než jazyky kompilované nebo jazyky s virtuálním strojem. Díky virtuálním strojům, jako je např. HHVM od vývojářů z Facebooku[7] zrychluje chod aplikace až devětkrát oproti PHP bez HHVM.

Výhody:

- velice podobná syntaxe jazyku C (tedy vhodné pro začátečníky),
- obrovská komunita po celém světě (již mnoho hotových řešení),
- multiplatformní,
- podpora téměř u všech poskytovatelů webhostingu.

Nevýhody:

- ne vše je objekt, jako například v Ruby,
- není vhodný pro desktopové aplikace, neboť knihovny pro tvorbu těchto aplikací nejsou na dostatečné úrovni,
- chybí ladící nástroj, který by byl přímo integrovaný v některém IDE.

2.2 Python

Python je objektově orientovaný skriptovací jazyk. Vznikl z jazyků Perl, Smalltalk, Tcl, a to v roce 1991. Jeho autorem je Guido van Rossum. Python je, z mnou vybraných skriptovacích jazyků, méně populární než PHP, ale i tak je v první desítce nejpoužívanějších jazyků. Je určen pro všechny typy aplikací i platforem. Nejnovější verze jazyka Python je 3.3.4 (9. 2. 2014).

Výhody:

- rychlý (oproti PHP) [8],
- jeden z nejvhodnějších jazyků pro začátečníky,
- multiplatformní.

Nevýhody:

- nedostatečná podpora webhostingů v České republice.

2.3 Ruby

Jazyk Ruby je dalším skriptovacím jazykem v mé bakalářské práci. Je oproti předešlým jazykům plně objektově orientovaný. Tento jazyk vznikl v roce 1995 a vytvořil jej Yukihiro Matsumoto pro vlastní potřebu. Ruby bylo používáno mnoho let převážně v Japonsku, odkud Matsumoto pocházel. Jakmile byla dokončena dokumentace v jiném jazyce než japonštině, Ruby začalo expandovat i do světa. Ruby, podobně jako Python, vychází z jazyků Perl a Smalltalk. Jako mnou poslední vybraný skriptovací jazyk je nejméně užívaný, ale přesto se v poslední době hojně rozšířil, a to převážně v zahraničí. V České republice natolik rozšířený není. Lze ho využít jak u malých, tak u velkých webových projektů, ale i u desktopových aplikací. Nejnovější verze Ruby je 2.1.1, vydána v únoru 2014.

Výhody:

- jednoduchá syntaxe,
- vhodný pro začátečníky.

Nevýhody:

- nedostatek české dokumentace,
- pomalejší než Python či PHP,
- nedostatečná podpora webhostingů.

2.4 C#

C# je nejmladší z mnou vybraných programovacích jazyků. Vznikl v roce 2002 společně s .NET frameworkem a vyvinula jej firma Microsoft. Jedná se o objektově orientovaný jazyk, který vychází z jazyků Java a C++. Co se týče jeho popularity, je méně populární než výše zmíněné PHP a Java, avšak je to na druhou stranu mladý jazyk a své příznivce si hledá. Stejně jako Java, je vhodný jak pro internetové, tak i desktopové aplikace. Mimo jiné je z něj tvořeno jádro operačních systémů Cosmos. Firma Microsoft ho také využívá při tvorbě svých operačních systémů, ale pro jádro je využíván jazyk C [9]. Nejnovější verzí C# je podle stránek firmy Microsoft verze 5.0 (srpen 2012) a .NETu 4.5.1 (říjen 2013).

Výhody:

- snadný vývoj aplikací.

Nevýhody:

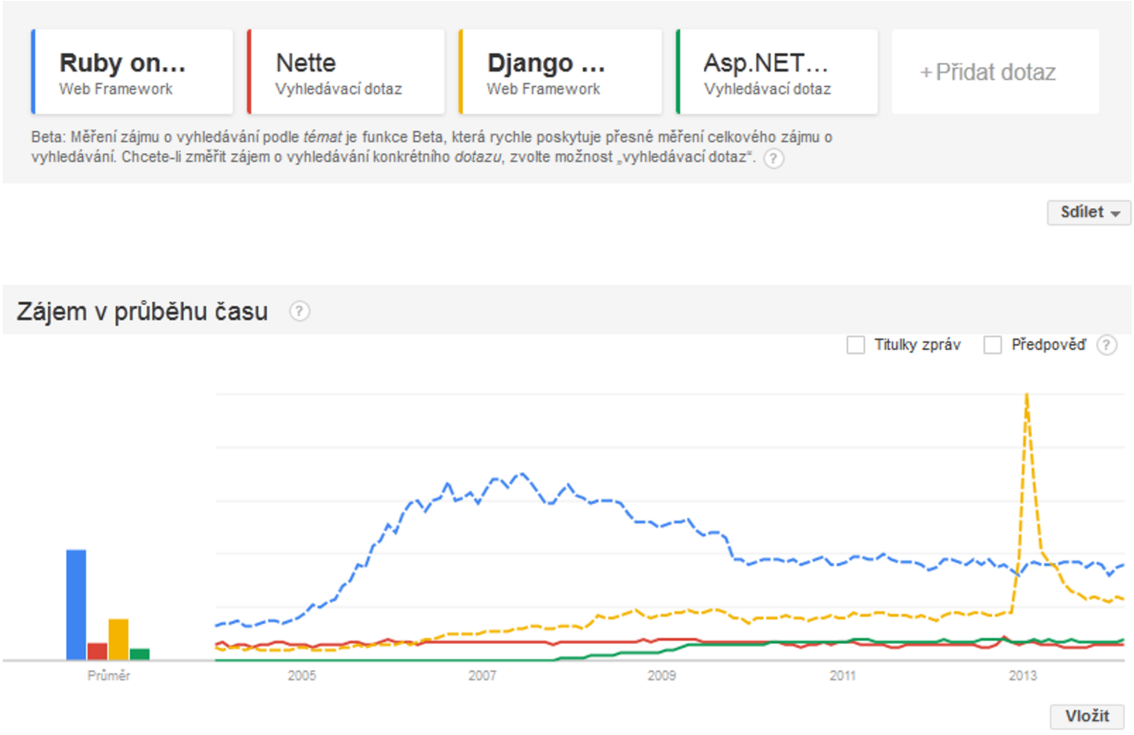
- pomalejší oproti C++,
- nedostatečná podpora webhostingů.

3 Použité frameworky a jejich popis

Pro svou bakalářskou práci jsem si vybral následující frameworky:

- Nette Framework,
- Django,
- Ruby on Rails,
- ASP.NET MVC.

Ze zmíněných frameworků jsem zatím pracoval okrajově pouze s Djangem a Nette Frameworkem. Výsledkem mé práce s Djangem byl malý informační systém pro hodnocení na základní škole. Django jsem si vybral i proto, že je to snad nejznámější framework pro Python, a jak ukazuje graf z Google Trends na obrázku 3.1, je o něj velký zájem. V Nette jsem psal redakční systém. Nette mě zaujalo ihned, když jsem vybíral PHP framework pro svou práci. V Česku je to nejvíce oblíbený framework ze čtyř mnou porovnávaných, ve světě se naopak takové popularity netěší. Ruby on Rails jsem si zvolil, protože frameworků pro Ruby moc není a Rails mě zaujal nejvíce. A ASP.NET MVC jsem si vybral, protože mě žádný jiný C# framework nezaujal natolik, jako ASP.NET MVC. Také proto, že je přímo součástí Microsoft Visual Studia.



Obrázek 3.1: Vyhledávanost frameworků ve vyhledávači Google[10].

Frameworky dbají na následující principy a techniky:

- KISS (Keep it simple, stupid) – v překladu „Zachovej to jednoduché, troubo“ je princip, který vede programátora k tomu, aby vzniklá aplikace zůstala jednoduchá a udržovatelná, než aby byla komplexní a neudržovatelná.
- DRY (Don‘t repeat yourself) – v přímém překladu „Neopakuj se“ je princip, který vede ke znovupoužívání kódu.
- AJAX (Asynchronous Javascript and XML) – jak uvádí W3C[11], AJAX není nový programovací jazyk, ale nová cesta k užívání existujících standardů. AJAX zajišťuje výměnu dat se serverem, aniž by znovu načítal celou stránku.
- ORM (Object-Relational Mapping) – programovací technika, která převádí data mezi typovými systémy, což zajišťuje jejich koexistenci mezi databázemi a objektově orientovanými jazyky.[12]
- Dependency Injection – není ani tak technika nebo princip, jak návrhový vzor, který snižuje závislosti mezi jednotlivými částmi programu. Výhodou je větší kontrola nad objekty, také dbá na znovupoužitelnost.[13]

3.1 Nette Framework

Nette framework je český framework pro vývoj webových aplikací, který vznikl v roce 2004 a veřejnosti byl představen až v roce 2007. Nyní je Nette k dostání v nejnovější verzi 2.1.1 pod licencí NewBSD a GPL. Hlavním vývojářem toho frameworku je David Grudl.

Nette framework je naprogramovaný v PHP 5 a plně využívá možností objektově orientovaného programování. Jak jsem již zmínil, Nette je český framework, tím pádem má v České republice snad největší programátorskou komunitu. Ve světě však Nette není natolik populární jako např. Zend Framework nebo CodeIgniter.

Proč používat Nette? Nette poskytuje, dle jeho internetových stránek[14], exceletní šablonovací systém, bezkonkurenční ladící nástroje, důmyslné zabezpečení před zranitelností, podporu HTML5, AJAX a SEO, efektivní databázovou vrstvu, atd. Dále dbá na principy jako je KISS, DRY, AJAX, Dependency Injection.

Nette framework se hodí jak pro jednoduché (malé www stránky), tak i pro složitější aplikace (např. blogy, e-shopy,. . .).

Jako šablonovací systém využívá Nette vlastního systému Latte. O Latte se zmíním v již samotném porovnávání frameworků.

Nevýhodou Nette je podle mě to, že se s ním člověk příliš neuplatní v zahraničí, protože není natolik populární. A za další nevýhodu lze považovat zpětnou kompatibilitu.

V Česku firmy vyvíjejí, alespoň podle inzerátů, na dvou frameworcích: Nette a Zend. Je to takový boj Davida s Goliášem. Zend má totiž velkou uživatelskou základnu po celém světě, ale Nette má, i přesto, že je populární pouze v České republice, co nabídnout.

Nette používají aplikace jako CSFD.cz, Slevomat, Mladá fronta a další.

3.2 Django

Django vyšlo poprvé na veřejnost v roce 2005 pod licencí BSD. Zakladatelem Djanga je firma Lawrence Journal-World, která jej prvně použila v roce 2003 pro správu zpravodajských stránek. Bylo pojmenováno po jazzovém kytaristovi Djangu Reinhardtovi.[15] Dnes jej používají takové stránky, jakými jsou Mozilla, Instagram, Pinterest a těší se velké popularity po celém světě. Nyní Django spravuje Django Software Foundation a jeho aktuální verze je 1.6.

Největší výhodu Djanga vidím v automaticky generované administraci, která se vytváří podle Modelu. Takže Django šetří čas dvakrát – tím, že je to framework a tím, že generuje administraci. Také komunita okolo Djanga není malá, na téma Django je také vydáno několik knih.

Jak jsem již naznačil, Django bylo prvně užito v novinách, tím pádem se také nejvíce ze všech frameworků hodí pro tzv. „obsahové“ aplikace. Samozřejmě ho lze použít na všechny typy aplikací.

Django se podobně jako Nette drží principu DRY, používá ORM přístup a implementuje MVC, resp. MTV(Model-Template-View).[16][17]

Django běží například na sociální síti Instagram nebo Pinterest.

3.3 Ruby on Rails

První verze frameworku Ruby on Rails, zkráceně Rails, byla vydána roku 2004 pod licencí MIT. Za frameworkem stojí dánský programátor David Heinemeier Hansson. Nejnovější verzí tohoto frameworku je 4.0.3, vydána 18 února 2014.

Mezi největší výhody Rails patří rychlost jeho vývoje, protože Rails se vyvíjí zároveň se samotným jazykem Ruby. Pro začátečníky není obzvláště těžký, sám jsem nad tím, abych mu porozuměl, nestrávil mnoho času. Nevýhodou, a to obrovskou, je malá podpora českých webhostingů. Nejlevnější hosting je 600 Kč za rok, průměrně se však cena pohybuje okolo 1200 Kč za rok. Co se týče webhostingů zdarma, tak ten je v ČR pouze jeden.

Framework Rails je převážně používán pro tvorbu webových aplikací. Mimo jiné na tomto frameworku běží aplikace jako Twitter či GitHub.

Rails implemetuje MVC, pracuje s různými typy databází, a to i (Oracle či MS SQL Server). Zastává jak princip DRY, tak i CoC.[18]

3.4 ASP.NET MVC

ASP.NET MVC je framework od společnosti Microsoft a je nejmladším frameworkem, který porovnávám ve své práci. Byl zveřejněn v dubnu 2009 pod licencí Microsoft Public Licence (MS-PL). Nejnovější verze je ASP.NET MVC 5 a byla vydána 13. října 2013. Má sloužit jako alternativa k ASP.NET WebForms.

Lze jej použít pro všechny typy aplikací. Problém nastává, když chceme aplikaci nahrát na hosting. Hostingů pro ASP.NET je velmi málo, o freehostingu nemluvě. V tom vidím velkou nevýhodu oproti všem frameworkům napsaných v jazyce Java či PHP.

Šablonovacích systémů je více, ale nejpoužívanějšími jsou následující dva:

- Razor View Engine – .cshtml nebo .vbhtml – podle použitého programovacího jazyka,
- Web Forms - .aspx.

Kromě vzoru MVC používá vzor Front Controller a dbá na principy DRY[19], Dependency Injection[20].

Výhodou ASP.NET MVC je, že i když uplynuly 4 roky po vydání tohoto frameworku, má již velkou uživatelskou bázi jak v České republlice, tak ve světě a dobře zpracovanou dokumentaci a tutoriály.

Tento framework využívají převážně aplikace firmy Microsoft, například Bing nebo MSDN.

4 Co se bude porovnávat

Kapitola okrajově popíše, co se bude konkrétně porovnávat mezi šablonovacími systémy již dříve zmíněných frameworků. Na úvod se vždy představí šablonovací systém spolu s ukázkou kódu v šabloně. Šablona bude u všech čtyř systémů vykreslovat stejnou HTML šablonu. Mimo jiné se v úvodu kapitola zaměří i na podporu šablonovacích systémů ve známých IDE (konkrétně na zvýraznění syntaxe, našeptávání), strukturu šablon (dědičnost) a značky, pomocí kterých se struktura šablon vytváří. Při uvedení příkladu dědění šablon se zaměřím na příklad dědění z výchozí šablony pro konkrétní podšablonu. Porovnání je pak rozděleno dále na pět částí. Na konci každé části se vždy ohodnotí všechny frameworky školní metodou (1 – bezproblémové, 5 – velice problémové). Kromě jiného jsou pro každý framework vytvořeny testovací aplikace, což zajišťuje i objektivní názor. Tyto testovací aplikace slouží jako zdroj pro porovnání šablonovacích systémů.

4.1 Lokalizace

V dnešní době, kdy na angličtinu můžeme narazit na každém rohu, se ani internetové aplikace neobejdou bez jazykových mutací. Snad každý framework podporuje lokalizaci, a není proto nutné vytvářet ty samé stránky v jiném jazyce. Nejde pouze o překlad textů, ale také o podporu pro formát data a času, či měny. Tato část nám ukáže, jak se který šablonovací systém vypořádává s překladem šablon, a která knihovna z frameworku je k tomu užita.

4.2 Testovatelnost

Testování šablon je kamenem úrazu u všech frameworků. Testovat lze téměř vše, šablony však nikoli. Nástroje pro testování jako Selenium ve frameworku zkrátka nejsou. Tato kapitola bude tedy zaměřena na to, zdali lze testování řešit pomocí plug-inu, nebo pouze externím programem, jako je například již zmíněné Selenium.

4.3 Bezpečnost

Bezpečnost internetových aplikací je snad jedno z nejprobíranějších témat dnešní doby. Bezpečnostních útoků je mnoho, práce se však bude zabývat pouze jedním, a to útokem XSS (Cross Site Scripting).

Útok XSS podstrčí škodlivý kód (většinou javascriptový) do kódu stránky aplikace. Většinou jde o vyskakovací okna nebo změny obrázků, avšak díky šikovným hotovým skriptům můžeme XSS využít např. jako keylogger.

Bránit se proti XSS lze pouze tím, že všechny řetězce ošetříme. Ošetření probíhá tzv. escapováním, které převádí znaky, které mají speciální význam v HTML (např. “, <, > atd.). Tato část srovnání se bude zabývat, jak si šablonovací systémy s escapováním vedou.

4.4 Rozšiřitelnost

Rozšiřitelností je v tomto porovnání myšleno převážně to, jak si můžeme v konkrétním šablonovacím systému ušetřit práci. Srovnání bude tedy zaměřeno na helpery. U Nette budou zmíněny i makra, protože je Nette, jako jediné, obsahuje. Tyto makra však nebudou zahrnuty do srovnání, aby nebyly znevýhodněny ostatní srovnávané frameworky .

Helpery jsou pomocné funkce, které nám pomáhají formátovat nebo upravovat data. Např. formátování data, zákaz escapování. Nejvíce helperů se vztahuje na textové řetězce (odstranění HTML tagů, záměna některých znaků v řetězci,. . .) či data nebo čísla.

Dále se zaměřím i na tvorbu vlastních helperů, protože ne všechno, co bychom potřebovali, výchozí helpery umí.

4.5 Práce s daty

Poslední část porovnání se bude věnovat přístupem k datům a práci s nimi. V šablonovacích systémech se tím myslí, jak získáme data z modelu, a poté je vypíšeme. Bude demonstrováno, pomocí jaké konvence (tečková, šipková,. . .) se přístup řeší. S výpisem většího počtu dat nerozlučně souvisí cykly, proto se zaměřím i na konvence pro psaní těchto cyklů.

5 Porovnání šablonovacích systémů

5.1 Základní informace

5.1.1 Nette

Framework Nette používá šablonovací systém pojmenovaný Latte, který se hodně podobá šablonovacímu frameworku Smarty, jež je také určen pro PHP. Všechny funkce frameworku, které nejsou součástí jazyka HTML, se zapisují do tzv. maker ({*výraz*}).

Podpora Nette, konkrétně Latte, je v různých IDE poměrně malá. Z volně dostupných je pro Nette nejvíce používané IDE NetBeans s pluginem Nette2 Framework, z placených potom PHPStorm nebo Sublime Text 2. NetBeans byl použit i pro testovací aplikaci, umí zvýraznit syntaxi, a také našeptávat po zadání prvního písmena. Sublime podporuje obě základní funkce také, tedy pokud se nainstaluje balíček pro Nette, PHPStorm, i přes mnoho vychytávek, podporu pro Latte nemá, i když je plugin již ve vývoji.

V Latte funguje dědičnost šablon pomocí makra {*block*}, případně {*include*}. V Latte se výchozí šablona zpravidla jmenuje @layout.latte (dále uvedena jako rodičovská šablona). Makro {*block*} nebo {*include*} nám pak označuje části rodičovské šablony, které se mění v závislosti na konkrétní šabloně potomka (např. HTML tag <*title*>). Poté v šabloně potomka pomocí makra {*block*} požadované části z @layout.latte nadefinujeme.

```
<!DOCTYPE html>
<html>
<head>
    <title>{block title|striptags}{/block}</title>
</head>

<body>
    {include content}
</body>
</html>
```

Obrázek 5.1: Rodičovská šablona Nette.

Jak jsem již zmínil, v Latte se rozlišují dva typy maker pro připojování obsahu {*block*} a {*include*}. V ukázce rodičovské šablony na obrázku 5.1 vidíme, že je užito obou. V prvním případě je pro HTML tag <*title*> použito makro {*block*}, tím říkáme, že daný obsah může, ale i nemusí být definovaný v šabloně potomka. Naopak v případě použití {*include*} je nutné, aby šablona potomka tento blok definovala. Pokud rodičovská šablona blok v šabloně potomka nenajde, nahlásí chybu.

V ukázce šablony potomka na obrázku 5.2 můžeme vidět použití makra {*block content*} pro předávání obsahu do rodičovské šablony a {*block title*} pro předání aktuální titulku stránky. Dále je zde použito n:makro, které bude zmíněno v podkapitole 5.5. Jako poslední se v ukázce nachází výpis článků z modelu pomocí cyklu foreach. O výpisu dat a práci s daty pojednává podkapitola 5.6. Pokud by tedy nebyl blok s názvem title definován, nic se nestane a do značky <*title*> se předá adresa URL aktuální stránky.

```
{block content}

{block title}<h1>Test CMS</h1>{/block title}
<a n:href="Clanek:novy">Napsat nový článek</a><br />

{foreach $clanky as $clanek}
    <div class="post">
        <h2><a href="{link Clanek:show $clanek->id}">
        {$clanek->nadpis}</a></h2>
        <div class="text">{$clanek->text}</div>
    </div>
{/foreach}
```

Obrázek 5.2: Šablona potomka Nette.

5.1.2 Django

Šablonovací systém Djanga je podobný šablonovacímu systému Smarty. V šablonách se vyskytují kromě HTML kódu jak proměnné ({{*proměnná*}}), tak tzv. tagy ({% *if podmínka* %}), ve kterých se vyskytují cykly, podmínky, ale také filtry, které se dají označit jako helpery v ostatních porovnávaných systémech.

Podpora Djanga ve zdarma dostupných IDE je obrovská, patří mezi ně např.: Aptana Studio, Eclipse nebo VIM, ke kterým je nutno doinstalovat PyDev plugin. Všechny editory umí díky PyDev pluginu zvýraznit syntaxi šablon a našeptávat. Z placených podporuje Django například PyCharm, který umí zvýrazňovat syntaxi v šablonách i našeptávat.

Dědičnost se v šablonovacím systému Djanga řeší obdobně jako v Latte pomocí tagů {% *block* %}, {% *endblock* %} a {*extends*}. Mezi tagy block se umístí proměnný obsah stránek. Je nutné si však vytvořit rodičovskou šablonu, a na tu se v šabloně potomka odkazovat pomocí tagu {*extends*}.

```
<html>
    <head>
        <title>{% block title %} {% endblock %}</title>
    </head>

    <body>
        {% block body %}
        {% endblock %}
    </body>
</html>
```

Obrázek 5.3: Rodičovská šablona Django.

Na obrázku 5.3 je názorně ukázáno, jak definujeme oblast vykreslení obsahu předaného z potomka. Vidět je i podobnost šablonovacího systému Djanga s Latte, protože oba vycházejí ze Smarty. Bloky v rodiči je nutno si pojmenovat a ukončit pomocí tagu {% *endblock* %}.

```
{% extends "base.html" %}
{%block body %}
<h1>{% block title %}Test CMS{% endblock %}</h1>

<a href= "novy">Napsat nový článek</a>
{% for clanek in clanky %}
    <div class="post">
        <h2><a href="show/{{clanek.id}}/">{{clanek.nazev}}
        </a></h2>
        <div class="text">{{clanek.text}}</div>
    </div>
{% endfor %}
{%endblock%}
```

Obrázek 5.4: Šablona potomka Django

Obrázek 5.4 demonstruje určení šablony, ze které potomek dědí pomocí tagu {% *extends* %}. Dále následuje definice počátku bloku „body“. Nadpis „Test CMS“ byl zároveň označen jako titulek stránky pomocí tagu {% *block title* %}. Dále následuje odkaz na vytvoření článku, cyklus for, ve kterém se vypisují jednotlivé nadpisy a texty článků. Většina tagů v Djangu má i uzavírací tagy a na ty není dobré zapomenout.

5.1.3 Ruby on Rails

V Ruby on Rails je možno používat mnoho šablonovacích systémů[21], v této práci se budu zabývat výchozím šablonovacím systémem ERB (Embedded Ruby). Všechen kód, který není HTML, je ohraničen značkami <%= *kód* %>. Mezi těmito značkami můžeme tedy nalézt kontrolní bloky nebo již dříve zmíněné helpery.

Podpora Ruby on Rails v IDE je docela velká, i když větší z nich potřebuje doinstalovat pluginy. Mezi IDE podporující Rails patří Aptana Studio 3, VIM a z placených například Komodo nebo RubyMine. Pro ukázkovou aplikaci bylo použito Aptana Studio, které poskytuje pouze zvýraznění syntaxe. VIM s doinstalovaným pluginem pak zvýrazňuje syntaxi a doplňuje text na základě kontextu. RubyMine i Komodo taktéž poskytují obě základní funkce. Komodo přímo automaticky doplňuje vhodné slova podle kontextu, než že by našeptávalo.

Dědičnost šablon je stejně jako ve dvou předchozích šablonovacích systémech podporována. Jako výchozí (dále rodičovská) šablona je zde definována application.html.erb umístěna ve složce views/layouts. Tato šablona je pak užita pro všechny stránky, pokud ovšem není řečeno jinak[1]. V Ruby on Rails se předá obsah, který se mění dle konkrétní stránky pomocí <%= *yield* %> a metody *content_for*.

V ukázce rodičovské šablony na obrázku 5.5 můžeme vidět, že pro stanovení měnících se částí šablony bylo použito <%= *yield* %> . Pomocí dvojtečky můžeme tuto požadovanou část pojmenovat. Pokud rodičovská šablona část s názvem title nenajde, tak se ve značce <*title*> objeví URL adresa stránky.

[1] Tato možnost se však nastavuje v controlleru.

```
<!DOCTYPE html>
<html>
<head>
  <title><%= yield :title %></title>
</head>
<body>
    <%= yield %>
</body>
</html>
```

Obrázek 5.5: Rodičovská šablona Ruby on Rails.

```
<h1><%= content_for :title, "Test CMS"%> Test CMS</h1>
<%= link_to 'Napsat nový článek', new_article_path %>
<% @articles.each do |article| %>
    <div class="post">
        <h2><%= link_to article.nadpis, article %></h2>
        <div class="text"><%= article.kratky_text %></div>
    </div>
<% end %>
```

Obrázek 5.6: Šablona potomka Ruby on Rails.

Na obrázku 5.6 lze vidět příklad předání parametru title pro rodičovskou šablonu pomocí metody *content_for*. Následně je zobrazena ukázka práce s helperem pro vytvoření odkazu. Poslední blok kódu je cyklus each do, který je v jazyce Ruby hojně užíván. V cyklu se nám opět vypíše nadpis článku, který je zároveň odkazem, a text článku.

5.1.4 ASP.NET MVC

Ve frameworku ASP.NET je možno vybírat mezi několika šablonovacími systémy. Mezi ty nejvíce používané a ty, které dává na výběr Microsoft Visual Studio jsou ASPX (převážně používaný pro WebForms) a Razor. V této práci se budu zabývat šablonovacím systémem Razor. V Razoru se všechen obsah jiný než HTML píše za @.

Dědičnost šablon je zde řešena způsobem, který se mi zdál ze všech nejsnadnější. Pomocí dynamické proměnné *ViewBag* můžeme předávat měnící se prvky HTML kódu v potomku tak, že výrazu (např. Title) nastavíme hodnotu. Samotný obsah stránky, který patří mezi tagy *<body></body>* se vykresluje pomocí metody *RenderBody()*. Výchozí (rodičovskou) šablonou pro Razor je _Layout.cshtml.

Nejlepší IDE pro tento šablonovací systém a framework celkově je Microsoft Visual Studio, existuje i několik jiných IDE, jako MonoDevelop nebo SharpDevelop, ale pravdou je, že se mi s Visual Studiem pracovalo nejlépe (našeptává, zvýrazňuje syntax). SharpDevelop zvýrazňuje pouze syntaxi. O MonoDevelop by se dalo říci, že je to zdařilá kopie Visual Studia. Nevýhodou je, že umí pracovat pouze s ASP.NET MVC 3.

V rodičovské šabloně, která je znázorněna na obrázku 5.7, je tedy nadefinováno pomocí proměnné Viewbag s výrazem Title místo, kam se bude předávat tato proměnná naplněna

```
<!DOCTYPE html>
<html>
<head>
    <title>@ViewBag.Title</title>
</head>
<body>
    <ul class="nav">
            <li><a href="/">Domů</a></li>
            <li><a href="/Contact/">Kontakt</a></li>
        </ul>
    @RenderBody()
</body>
</html>
```

Obrázek 5.7: Rodičovská šablona ASP.NET MVC

v šabloně potomka. Metoda *RenderBody* poté vykreslí všechen HTML obsah předaný šablonou potomka.

```
@model IEnumerable<CMSDemo.Models.Article>
@{
    ViewBag.Title = "Test CMS";
}

<h1>@ViewBag.Title</h1>
@Html.ActionLink("Napsat nový článek", "../Administration/Create")
@foreach (var c in Model)
{
    <div class="post">
        <h2><a href="/Article/Show?id=@c.ArticleId">@c.Nadpis</a></h2>
        <div class="text">@c.KratkyText</div>
    </div>
}
```

Obrázek 5.8: Šablona potomka ASP.NET MVC

První řádek kódu na obrázku 5.8 specifikuje, jakého typu má být očekávaný model. Na dalším řádku je v konstrukci @{} naplněna proměnná *ViewBag* s výrazem Title na hodnotu „Test CMS“. Dále lze vidět, že jakýkoli výraz z proměnné ViewBag může být použit i v šabloně potomka, jak bylo učiněno v tvorbě nadpisu. Poté je v šabloně použit helper pro tvorbu odkazu. A jako poslední, cyklus foreach pro výpis článků.

	Nette	Django	Rails	MVC
Podpora v IDE	2	1	2	3
Složitost tvorby dědičnosti	1	1	1	1
Známka	1,5	1	1,5	2

Tabulka 5.1: Výsledky v kategorii ”Základní informace”

5.2 Lokalizace

5.2.1 Nette

Ačkoli Nette lokalizace podporuje, nedisponuje úplným řešením, jak je používat. V Nette je pro ně vytvořeno rozhraní ITranslator. Abychom mohli texty překládat, stačí vytvořit překladač, který implementuje rozhraní ITranslator. Samotný překladač si musí programátor vytvořit sám, což je na jednu stranu výhodné, protože to nechává programátorovi volnou ruku, ale na stranu druhou nám tím framework přidává práce.

Jako možnou implementaci překladače je nejvhodnější použít buďto gettext, nebo metodu asociativního pole (Zend_Translator) s tím, že metody s asociativním polem jsou omezeny na počet položek pro překlad. Gettext je rychlejší, udržuje totiž data v mezipaměti, na druhou stranu se hůře spravuje (čte), neboť ukládá data do binárního souboru. Nabízí se i jiné možnosti, např. XML soubor nebo relační databáze.

Druhým řešením je možnost použít plugin. Například Gettext Translator, který má velice podrobný návod a snadno se používá. Dalším pluginem, který umí překládat šablony a formuláře je TemplateTranslator. Pro správu překladů je vhodný např. Nette Translation Panel[2].

Samotný překlad v šablonách se na konkrétní implementaci překladače příliš neliší. Vše je nutné nakonfigurovat a v samotných šablonách vypadá jako na obrázku 5.9.

```
<h1 n:block="title">{_'Contact'}</h1>
```

Obrázek 5.9: Ukázka překladu šablon v Nette

5.2.2 Django

Oproti Nette má Django lokalizaci plně zahrnutou ve svých dispozicích a programátor nemusí téměř nic dopisovat. Pro lokalizace je použit nástroj Gettext, a i když je to externí nástroj, jeho instalace (obzvláště na Linuxu) není nic složitého.

Samozřejmě jako u všech porovnávaných frameworků je nutné nastavit pár maličkostí, než se lokalizace začne tvořit. U Djanga je třeba určit v souboru settings.py, jaký jazyk se má použít, jaké jsou podporovány jazyky, zdali chceme mít zapnutou (či vypnutou lokalizaci), a kde se soubory s lokalizací nacházejí. Dalším krokem je samotné vyznačení textu v šabloně, který se má překládat pomocí tagu {% *trans "Klíčové slovo"* %}. V šabloně je také na samém začátku zapotřebí načíst modul pro internacionalizaci {% *load i18n* %}. Toto nastavení demonstruje obrázek 5.10.

Jako soubory pro překlad používá Django dva typy souborů: textový (.po) a binární (.mo). V textovém souboru se vytvoří překlady, které budeme v aplikaci používat a tyto překlady se poté přeloží do binárního souboru. V textovém souboru jsou vždy obsaženy dvojice msgid a msgstr, kde msgid je námi překládané slovo a msgstr je samotný překlad tohoto slova. Názornou ukázku souboru pro překlad lze vidět na obrázku. 5.11

[2]Nette Translation Panel, https://github.com/jsmitka/Nette-Translation-Panel

```
{% load i18n %}

<h1>
    {% block title %}{%trans "Kontakt" %}{% endblock %}
</h1>
```

Obrázek 5.10: Ukázka překladu šablon v Djangu

```
#: templates/kontakt.html:5
msgid "Kontakt"
msgstr "Contact"
```

Obrázek 5.11: Ukázka překladových souborů v Djangu

5.2.3 Ruby on Rails

V Ruby on Rails je překlad šablon jednodušší než v Nette. Pro lokalizace používá vlastní API I18n. Je však možné užít, stejně jako v Nette, relační databázi nebo gettext. Práce se však bude zabývat lokalizací pomocí API I18n.

Před samotným překladem šablon je nutné mít I18n správně nastavenou. O polovinu se postará generátor projektu, stačí pouze odkomentovat dva řádky. O druhou polovinu se musí postarat programátor, který vytvoří funkci pro nastavení aktuálního jazyka a funkci pro vytváření URL adres. Právě díky URL adresám je jednoduché zjistit, jaký je požadovaný jazyk. Práce se nebude zabývat nastavováním lokalizací, ale spíše jejich použití v šablonách.

Rails používá pro uložení překladů soubory s příponou .yml. V tomto souboru si nadefinujeme podle cesty k šabloně klíčová slova, na která se budeme odkazovat. Příklad tohoto souboru je demonstrován na obrázku 5.12.

```
en:
  contact:
    index:
      contact: "Contact"
```

Obrázek 5.12: Ukázka překladových souborů v Ruby on Rails

```
<h1><%=t ('.contact')%></h1>
```

Obrázek 5.13: Ukázka překladání šablon v Ruby on Rails

V souboru cs.yml je nadefinováno, že se prvek s názvem contact bude překládat na Kontakt v případě, že požadovaný jazyk je čeština a v případě, že bude jazykem angličtina, bude se překládat na Contact. V šabloně se potom díky metodě *I18n.translate*, nebo spíše díky její pomocné metodě *t*, odkazuje na konkrétní prvek ze souboru cs.yml nebo en.yml. Demonstrace překladu je zobrazena na obrázku 5.13.

5.2.4 ASP.NET MVC

ASP.NET MVC zahrnuje nejsnadnější a nejpřehlednější způsob pro tvorbu lokalizací. Využívá se jmenný prostor *System.Globalization* a její jediná třída *CultureInfo*, její instance je poté využita v souboru Global.asax.cs pro nastavení aktuálního jazyka.

Pro samotné překlady se v ASP.NET MVC využívají resource soubory (.resx). Ukázka souboru je na obázku 5.15. Jejich plnění je velice snadné díky rozhraní, kde vyplňujeme pouze název a hodnotu. Nutné je však dodržovat pojmenování těchto souborů. První část názvu souboru je název samotný (pokud máme v aplikaci více View, není na škodu pojmenovávat resource soubory stejně jako samotná View), druhou část, která je oddělena tečkou, tvoří zkratka pro konkrétní jazyk překladu. I zde je nutno pojmenovávat jazyky zkratkami, které jsou již zažité (cs – český jazyk, en – anglický jazyk ...). Poslední věc, která je užitečná, je pojmenování jmenného prostoru ve vlastnostech souboru.

```
@using Microsoft.Security.Application
@{
    ViewBag.Title = Translate.Kontakt.contact;
}

<h1>@ViewBag.Title</h1>
```

Obrázek 5.14: Ukázka překládání šablon v ASP.NET MVC

V aplikaci se pak na tento soubor odkazujeme pomocí pojmenovaného jmenného prostoru např. Translate. Poté vybereme vhodný soubor s překladem (lze vidět, že zkratky jazyků se zde nepíší, šablonovací systém si aktuální jazyk zjistí sám). Jako poslední vybereme požadovaný název. Viz. příklad na obrázku 5.14.

Name	Value
contact	Contact

Obrázek 5.15: Ukázka překladových souborů v ASP.NET MVC

	Nette	Django	Rails	MVC
Podpora lokalizace	1	1	1	1
Složitost tvorby a nastavení lokalizace	3	2	1	1
Složitost tvorby lokalizačních souborů	2	1	2	1
Známka	2	1,3	1,3	1

Tabulka 5.2: Výsledky v kategorii "Lokalizace"

5.3 Testovatelnost

5.3.1 Nette

Automatizované testy šablon v Nette nemají podporu ani z malé části, i přes to, že disponuje poměrně jednoduchou testovací knihovnou Nette/Tester. Pluginy pro tuto problematiku také nebyly vytvořeny, a proto se většina programátorů programujících právě v Nette uchyluje k testování šablon pomocí Selenia nebo jiných nástrojů.

5.3.2 Django

Django jako jediné obsahuje nástroje pro testování šablon. Nejde sice o kompletní Selenium testy, ale na drobné testování šablon to bohatě postačí. Třída test client se chová jako prohlížeč a dovoluje nám sledovat a testovat chování šablon (resp. celé aplikace).

Mezi hlavní funkce test klienta patří simulace GET/POST požadavků a sledování odpovědí, sledovat řetězec přesměrování (kontrola správnosti URL adresy, stavové kódy v každém průběhu). Pro použití test klienta je nutné jej naimportovat z django.test.client.

5.3.3 Ruby on Rails

Rails je na tom s testováním šablon podobně jako Django, přímá podpora testování šablon zde sice neexistuje, ale pluginy ano[22]. Pro Rails se používají pluginy pro nástroje jako Selenium nebo Capybara. S externím užíváním některého testovacího nástroje nemá Rails problém.

5.3.4 ASP.NET MVC

Ani ASP.NET MVC nemá pro Razor speciální nástroj pro automatizované testy šablon. Avšak za velkou výhodu lze považovat to, že můžeme pomocí jmenného prostoru OpenQA Selenium používat a psát automatizované testy přímo ve Microsoft Visual Studiu. Mimo Selenium je také možné externě používat již zmíněný nástroj Capybara nebo Visual Studio Testing Tools.

	Nette	Django	Rails	MVC
Podpora testování šablon	5	3	5	5
Pluginy/externí programy pro testovaní šablon	5/1	5/1	1/1	3/1
Celkově	4	3	3	3,5

Tabulka 5.3: Výsledky v kategorii "Testovatelnost"

5.4 Bezpečnost

5.4.1 Nette

Nette si v ohledu bezpečnosti stojí velice dobře, obzvláště co se týče XSS útoků. Používá svou vlastní technologii zvanou Context-Aware Escaping, kterou mimo jiné používá i Google[23], a díky které se escapování provádí úplně samo. A není to vše, co tato technologie umí. Její hlavní předností je, že dokáže rozlišit, v jakém kontextu se daná proměnná nachází (může jít o HTML, JavaScript, CSS, apod.). Programátorovi stačí pouze použít makro pro vypsání proměnné {*$var*} a o zbytek se již postará výše zmíněná technologie. A v tom je silná stránka Nette v ohledu na bezpečnost. Ať už programátor na bezpečnost myslí nebo ne, Nette vždy zajistí, aby se zabránilo bezpečnostnímu útoku XSS.

Je dobré zmínit, že když programátor nechce mít escapování zapnuté, stačí místo makra {*var*} použít makro {*!var*} nebo použít modifikátor | *noescape*.

```
{$var = "<script> alert('XSS útok'); </script>"}

<p style="{$var}">Text</p>

{!$var}
```

Obrázek 5.16: Demonstrace užití zapnutého i vypnutého escapování v Nette

V ukázce na obrázku 5.16 je vytvořena proměnná *$var*, která by nám v běžné neošetřené aplikaci nebo při použití makra {*!$var*} zobrazila na obrazovku prohlížeče hlášku s textem ‚XSS útok'. Dále je zde uveden příklad, kdy tuto proměnou vkládáme do atributu style. Jako poslední se zde nachází ukázka výpisu neošetřené proměnné. Tímto chci i demonstrovat funkčnost Context-Aware Escaping. V případě, že máme escapování zapnuté, bude vypadat HTML kód stejně jako na obrázku 5.17.

```
&lt;script&gt; alert('XSS útok'); &lt;/script&gt;

<p style="\&lt;script\&gt; alert\('XSS útok\'\)\; \&lt;\/script\&gt;">Text</p>

<script> alert('XSS útok'); </script>
```

Obrázek 5.17: Výsledek demonstrace escapování na obrázku 5.16

V prvním případě Nette převedlo potencionálně škodné znaky na HTML entity a javascriptová hláška samozřejmě nebyla zobrazena. V druhém případě byla proměnná taktéž ošetřena a jde zde právě vidět rozdíl mezi prvním a druhým stylem escapování, kdy v prvním případě proměnnou pouze vypisujeme, v druhém ji vkládáme do atributu style a v tomto kontextu byla taktéž ošetřena. U posledního případu nalezneme výpis neošetřené proměnné.

5.4.2 Django

Django taktéž podporuje automatické escapování. Stačí vypsat proměnnou pomocí bloku {{*var*}}, kde *var* je proměnná, v tomhle případě, se škodlivým obsahem. Django však oproti předešlému Nette nepodporuje Context-Aware Escaping či jinou podobnou technologii.

I v Djangu se může escapování vypnout tím, že označíme proměnnou za bezpečnou. Lze to udělat dvěma způsoby. Buďto použitím filtru *safe* – {{ *var* | *safe* }}, nebo bloku:

{% *autoescape off* %}
  {{*var*}}
{% *endautoescape* %}

```
{{var}}

<p style="{{var}}">Text</p>

{{var | safe}}
```

Obrázek 5.18: Demonstrace užití zapnutého i vypnutého escapování v Djangu

Obrázek 5.18 demonstruje stejný případ jako u předešlého frameworku. Proměnná *var* obsahuje javascriptový kód s hláškou. V jednom případě proměnnou vypíšeme a v druhém ji vložíme do atributu style. V posledním vidíme ukázku výpisu v neošeřeném tvaru.

```
&lt;script&gt;alert(&#39;XSS Útok&#39;);&lt;/script&gt;<br>

<p style="&lt;script&gt;alert(&#39;XSS Útok&#39;);&lt;/script&gt;">Text</p>

<script>alert('XSS Útok');</script>
```

Obrázek 5.19: Výsledek demonstrace escapování na obrázku 5.18

Jak můžeme z kódu na obázku 5.19 poznat, Django escapovalo proměnnou vždy stejnou cestou – potencionálně škodné znaky převedlo na HTML entity. Důvodem, proč je výsledek stejný, je to, že Django nerozlišuje kontext. V testovací aplikaci nebo i jiných méně náročných aplikacích, však nerozeznání kontextu nehraje tak velkou roli. Avšak kdyby byla technologie na rozpoznání kontextu implementována (alespoň díky pluginu), nebylo by to v Djangu na škodu.

5.4.3 Ruby on Rails

Na úvod je dobré zmínit, že o Rails se říká, že je děravé jako cedník. A tohle tvrzení je v hodně případech bezpečnosti pravdivé. Vývojáři však tyto chyby opravují a bezpečnostních děr ubývá. Naštěstí se bezpečnostní chyby týkají spíše útoků typu SQL Injection než XSS útoků.

V dřívějších verzích Ruby on Rails bylo escapování podporováno, nebylo však automatické. V novějších verzích (tj. 3.x.x a výš) podpora automatického escapování funguje podobně jako v Djangu či ASP.NET MVC. Vše, co se vloží do bloku <%= % > je automaticky escapováno.

Existuje zde ještě jedna alternativní možnost, jak se bránit proti vkládání tagu *<script >* tam, kam nechceme. Jedná se o helper s názvem sanitize, který nám ve vybrané části HTML kódu ponechá ty tagy a atributy, které chceme (resp. které si nadefinujeme). Podpora rozeznávání kontextu při escapování zde stejně jako v Djangu nefunguje, ani pro ni neexistuje žádný plugin.

Ani v tomto frameworku není v případě nutnosti problém vypnutí escapování. Lze to provést třemi způsoby. Prvním, nejvíce používaným, je použití helperu *raw*, jako druhý způsob může být použita metoda *html_safe*, která je ekvivalentem k helperu raw a jako poslední způsob je dosti neobratný (myšleno v tomto kontextu) helper *content_tag*.

```
<%= @var = "<p>Text<script>alert('XSS Útok');</script></p>"%>

<p style="<%= @var %>">Text</p>

<%= raw @var%>
```

Obrázek 5.20: Demonstrace užití zapnutého i vypnutého escapování v Ruby on Rails

V ukázce, která je zobrazena na obrázku 5.20, vidíme jako v předchozích ukázkách výpis ošetřené proměnné *var* obsahující škodlivý javascriptový kód, dále uložení do stylu a výpis neošetřené proměnné.

```
&lt;script&gt;alert(&#39;XSS Útok&#39;);&lt;/script&gt;

<p style="&lt;script&gt;alert(&#39;XSS Útok&#39;);&lt;/script&gt;">Text</p>

<script>alert('XSS Útok');</script>
```

Obrázek 5.21: Výsledek demonstrace escapování na obrázku 5.20

Z obrázku 5.21 je tedy vidět, že Rails skutečně escapuje proměnnou vždy stejně, a stejně jako předešlé frameworky, nahradí potencionálně škodlivé znaky za HTML entity. Jak už bylo zmíněno, nerozeznává kontext, ale i tak je escapování v tomto případě úspěšné.

```
<%= @var = "<p>Text<script>alert('XSS Útok');</script></p>"%>

<%= sanitize @var, tags: %w(p)%>
```

Obrázek 5.22: Demonstrace užití helperu sanitize v Ruby on Rails

Obrázek 5.22 demonstruje použití helperu *sanitize*, kterému jsme nadefinovali, nad kterým řetězcem se má spustit, a jaké tagy a jejich obsah ponechat.

Výsledkem je řetězec, z kterého jsou odfiltrovány všechny tagy, které nebyly definovány. Tento řetězec se nachází na obrázku 5.23.

```
<p>Text</p>
```

Obrázek 5.23: Výsledek demonstrace užití helperu sanitize na obrázku 5.22

5.4.4 ASP.NET MVC

Zabezpečení proti XSS ve frameworku ASP.NET MVC je velice podobné jako v Djangu či Ruby. V Djangu bylo vše, co se nacházelo v bloku {{}}, automaticky escapováno, v ASP.NET MVC je vše, co se nachází za @ automaticky escapováno taktéž. Ani tento framework neimplementuje technologii Context-Aware Escaping, tudíž je vše escapováno do stejného výsledku.

Není problém jako u všech předešlých frameworků escapování vypnout. Dělá se to pomocí helperu *Html.Raw()*.

Výhodou je, že obranu proti XSS můžeme zvýšit pomocí knihovny AntiXSS[24]. Knihovna se jednoduše nainstaluje přes NuGet, poté se již odkazujeme přes jmenný prostor *Microsoft.Security.Application.Encoder* (původně bylo místo Encoder užíváno AntiXSS) na jednotlivé metody knihovny. AntiXSS dokáže escapovat řetězce jak pro použití v CSS, tak pro URL nebo XML.

Ve spojení s AntiXSS je ASP.NET MVC silným konkurentem Nette.

Další možností, jak se bránit proti XSS, je přímé odstraňování tagu *<script >* i celého jeho obsahu z požadovaného řetězce. Je to vlastně podobný princip jako v Ruby on Rails a jeho helperu *sanitize*. V ASP.NET MVC však nemusíme definovat tagy, které v řetězci nechceme, automaticky si škodlivé tagy rozpozná. Pro použití tohoto principu je nutné nainstalovat pomocí NuGet knihovnu HtmlSanitization a poté díky její mětodě *GetSafeHtmlFragment()* zabezpečovat jednotlivé řetězce.

```
@{ var text = "<script>alert('XSS Útok!');</script>"; }

@text <br />

<p style="@text">Text</p>

@Html.Raw(text)
```

Obrázek 5.24: Demonstrace užití zapnutého i vypnutého escapování v ASP.NET MVC

Jako v předchozích ukázkách, je použití escapování a jeho vypnutí demonstrováno na jednoduchém javascriptovém kódu zobrazeném na obrázku 5.24.

```
&lt;script&gt;alert(&#39;XSS &#218;tok!&#39;);&lt;/script&gt; <br />

<p style="&lt;script&gt;alert(&#39;XSS &#218;tok!&#39;);&lt;/script&gt;">Text</p>

<script>alert('XSS Útok!');</script>
```

Obrázek 5.25: Výsledek demonstrace escapování na obrázku 5.24

Výsledek escapování je zobrazen na obrázku 5.25. Po escapování je výstup totožný jak v případě samotného vypsání, tak použití v parametru style.

```
@{ var text = "<script>alert('XSS Útok!');</script>"; }

@Encoder.HtmlEncode(text)

<p style="@Encoder.CssEncode(text)">Text</p>
```

Obrázek 5.26: Demonstrace užití knihovny AntiXSS v ASP.NET MVC

Na obrázku 5.26 je demonstrováno použití knihovny AntiXSS. Změnilo se pouze to, že text vkládáme jako parametr do metody *HtmlEncode*, která převádí řetězec pro použití v HTML a do metody *CssEncode*, která převádí řetězec pro použití v CSS.

```
&amp;lt;script&amp;gt;alert(&amp;#39;XSS Útok!&amp;#39;);&amp;lt;/script&amp;gt;

<p style="\00003Cscript\00003Ealert\000028\000027XSS\000020\0000DAtok\000021\000
          027\000029\00003B\00003C\00002Fscript\00003E">Text</p>
```

Obrázek 5.27: Výsledek demonstrace užití knihovny AntiXSS na obrázku 5.26

Rozdíl mezi escapováním bez AntiXSS a s AntiXSS je zřejmý při prvním pohledu na obrázek 5.27.

```
@{ var text = "<script>alert('XSS Útok!');</script>"; }

@Sanitizer.GetSafeHtmlFragment("<p>"+text+"</p>") <br><br>
```

Obrázek 5.28: Demonstrace užití knihovny HtmlSanitization v ASP.NET MVC

Na obrázku 5.28 je demonstrováno použití knihovny HtmlSanitization a její metody *GetSafeHtmlFragment* na odstavec obsahující javascript.

```
&lt;p&gt;&lt;/p&gt;
```

Obrázek 5.29: Výsledek demonstrace užití knihovny HtmlSanitization 5.28

Obrázek 5.29 znázorňuje výsledek použití metody *GetSafeHtmlFragment* jejíž výsledkem je řetězec převeden na HTML entity a všechen obsah tagu *<script >* i tag samotný byl odtraněn.

5.5 Rozšiřitelnost

5.5.1 Nette

Nette zahrnuje necelých 30 helperů, které programátorovi usnadňují práci. Jedná se převážně o helpery pro práci s řetězci a jejich formátování. Nette mimo helpery používá taktéž

	Nette	Django	Rails	MVC
Účinnost escapování	1	1	1	1
Kontextové escapování	1	5	5	3
Jiné metody/knihovny pro ochranu před XSS	5	5	3	3
Celkově	2,3	3,6	3	2,3

Tabulka 5.4: Výsledky v kategorii "Bezpečnost"

makra. Makra by se mohly definovat jako značky, které zahrnují jak řídící konstrukce, tak značky pro tvorbu struktury stránky (include, layout, block) nebo vytváření proměnných či URL odkazů. V některých frameworcích existují helpery na tvorbu formulářů. V Nette je to trošku jinak, neboť pro tvorbu formulářů má speciální metodiku, kdy se formulář nevytváří v šabloně, ale v presenteru.

Použití helperu je velice jednoduché. Pro názornou demonstraci použití helperů budou ukázány tři nejvíce zajímavé nebo užitečné helpery (*webalize, number, bytes*).

```
{var $text = "Text o mně"}
{$text|webalize}<br /><br />

{var $cislo = 1024.1238}
{$cislo|number:3:',':'.'}<br /><br />
{$cislo|bytes}
```

Obrázek 5.30: Ukázka užití vybraných helperů v Nette

Na obrázku 5.30 je demonstrováno použití výše zmíněných helperů. Výsledkem použití helperu *webalize* bude převedení řetězce „Text o mně" na řetězec „text-o-mne", který se hodí pro tvorbu URL adresy. Další dva helpery jsou aplikovány na číselnou hodnotu. Helper *number* vypíše číslo 1024.1238 jako číslo 1.024,124. Je tedy patrné, co helper *number* dělá. Díky předaným parametrům (počet desetinných míst, znak pro oddělení celé a desetinné části a oddělení tisíců) vypíše číslo ve stanoveném formátu a zbytek zaokrouhlí. Helper *bytes* přepíše číslo do tzv. lidsky čitelné podoby, v tomto případě 1 kB.

Poslední částí, kterou se má rozšiřitelnost zabývat, je tvorba vlastních helperů. Tvorba helperů v Nette nečiní problémy. Jediné, co je za potřebí, je vytvořit funkci *createTemplate* (pokud již není vytvořená) v kterémkoliv presenteru, nejlépe však v BasePresenteru, odkud můžeme volat helper v jakékoliv části aplikace. Ve funkci *createTemplate* se poté zaregistruje helper pomocí metody *RegisterHelper*. Tato metoda přijímá dva parametry – samotný název helperu a anonymní funkci či callback. V šabloně se poté helper volá úplně stejně jako výchozí helpery. Ukázka vytvoření vlastního helperu a jeho volání v šabloně je znázorněna na obrázcích 5.31 a 5.32

```php
protected function createTemplate($class = NULL)
{
    $template = parent::createTemplate($class);
    $template->registerHelper('pozdrav',
        function ($name) {
            return 'Ahoj '.$name;
    });
    return $template;
}
```

Obrázek 5.31: Ukázka tvorby vlastního helperu v Nette

```
{var $jmeno = 'Pavle'}
<p>{$jmeno|pozdrav}</p>
```

Obrázek 5.32: Ukázka volání vlastního helperu v Nette

5.5.2 Django

Jediný framework, kde se pomocné funkce nenazývají helpery, ale filtry, je Django. Django disponuje filtry, jejichž počet je okolo 60, a to převážně pro práci s textem, což je dvojnásobný počet než u předchozího Nette. Podobně jako v Nette jsou makra, má i Django své tagy, které mají naprosto stejný význam jako v Nette. Poskytují řídící konstrukce, tagy pro rozvržení stránky (extends, block) apod. Ani zde se formuláře neskládají přímo v šabloně, ale ve speciálním souboru pro formuláře forms.py. Jak jsem již uvedl Django má mnoho filtrů, demonstrovat budu opět trojici nejzajímavějších, a to konkrétně – *truncatewords, timeuntil a floatformat*. Tuto demonstraci lze vidět na obrázku 5.33.

```
{{text|truncatewords:3}}

{{deadline|timeuntil}}

{{number|floatformat:2}}
```

Obrázek 5.33: Ukázka užití filtrů v Djangu

Po použití filtru *truncatewords* a jeho jediného paramteru (počet slov) se vypíše požadovaný počet slov z řetězce a zbytek bude nahrazen třemi tečkami. *Timeuntil* je zajímavý helper, který nám na základě předaného data spočítá, kolik zbývá od dnešního (výchozí) nebo daného data roků/měsícu/.../minut do data předaného. Poslední v ukázce je helper *floatformat*, který pracuje s čísly. Tento helper upraví číslo s desetinnou čárkou na požadovaný počet desetinných míst a zaokrouhlí jej.

Filtry si samozřejmě může programátor vytvořit sám a není na tom nic složitého. Nejprve je nutné vytvořit složku templatetags v kořenovém adresáři aplikace, v ní dva soubory (__init__.py a soubor s filtry (např. filtry.py)). Poté všude, kde chce programátor užívat své filtry, je nutné je načíst pomocí tagu {% *load filtry.py* %}. V souboru filtry.py

se naimportuje knihovna template a vytvoří se instance její metody *Library*, která se pojmenuje *register*. Nyní je vše nastaveno a stačí vytvořit filtr a zaregistrovat jej. Samotný filtr se tvoří stejně jako metoda, která přijímá jeden nebo dva parametry. Registrace filtru se provádí přes instanci metody *Library*, pomocí metody *filtr(název_filtru)*. Volání vlastního filtru je stejné jako volání některého z výchozích filtrů. Jak taková tvorba filtru a jeho následné volání vypadá, je znázorněno na obrázku 5.34 a 5.35.

```
from django import template

register = template.Library()

def pozdrav(name):
    return 'Ahoj'+name

register.filter('pozdrav', pozdrav)
```

Obrázek 5.34: Ukázka tvorby vlastního filtru v Djangu

```
{{"Pavle"|pozdrav}}
```

Obrázek 5.35: Ukázka volání vlastního filtru v Djangu

5.5.3 Ruby on Rails

Rails obsahuje největší počet helperů (přes 100), tento velký počet je dán také tím, že Rails obsahuje v helperech i takové, které jsou v jiných frameworcích považovány za makra / tagy / bloky. Nejobsáhlejšími skupinami helperů jsou ty, které pracují s daty, následující těmi, které pomáhají s tvorbou formulářů nebo helpery pro práci s textem.

Na místě je opět demonstrace použití některých z helperů. Vybrány byly dva pro práci s čísly (*number_to_currency* a *number_with_delimiter*) a jeden pro práci s daty (*select_day*).

```
<%= number_to_currency(55489875654.0859, :locale => :cs)%>

<%= number_to_human(12525552.1234, separator: ",", precision:5)%>

<%= select_day(5)%>
```

Obrázek 5.36: Ukázka užití helperů v Ruby on Rails

Na obrázku 5.36 je možné vidět ukázku použití výše zmíněných helperů. Helper *number_to_currency* má výstižný název a dělá přesně to, jak se jmenuje – převede číslo na měnu podle daného jazykového prostředí. Je nutné mít všechny potřebné informace pro měnu nastavené v souboru s lokalizacemi, jinak je jako výchozí jazykové prostředí nastavena Americká angličtina. Další helper, *number_to_human*, který již tak výstižný název nemá (převod čísla na člověka, je docela těžko představitelná věc), avšak co ztratil na výstižném názvu, nabyl na své funkčnosti. Tento helper tedy převede jakékoli číslo na

lidsky čitelnou podobu. Číslo uvedené v příkladu bude převedeno na „12,526 Million“, samozřejmě se zde mohou použít jednotky, lokalizace, atd... Poslední helper vytvoří HTML značku select s možnostmi pro výběr dne v aktuálním měsíci, pokud uvedeme v závorce číselný parametr (v příkladu číslo 5), tak bude vybrán jako výchozí tento den.

Vlastní helpery a jejich tvorba v Rails nečiní žádné problémy a je velice jednoduchá. Tvorbu vlastního helperu znázorňuje obrázek 5.37. Framework Rails vytvoří celou kostru projektu za nás i se složkou s helpery. Pokud chceme používat vlastní helper v celé aplikaci, je dobré jej vytvořit v souboru application_helper.rb. V tomto souboru vytvoříme novou metodu (helper), která bude mít své parametry. Volání vlastního helperu ve view je opět naprosto stejné, jako bychom volali jeden z výchozích helperů. Volání vlastního helperu lze vidět na obrázku 5.38.

```
module ApplicationHelper
  def pozdrav(name)
    return "Ahoj "+name
  end
end
```

Obrázek 5.37: Ukázka tvorby vlastního helperu v Ruby on Rails

```
<%= pozdrav "Pavle"%>
```

Obrázek 5.38: Ukázka volání vlastního helperu v Ruby on Rails

5.5.4 ASP.NET MVC

Co se ASP.NET MVC frameworku týče, ohledně helperů je oproti jiným frameworkům pozadu. Helperů je celkem 11 a většina z nich (10) jsou pro tvorbu formulářů (*RadioButton(), TextBox()*,...). Téměř jediným neformulářovým helperem je *ActionLink*. Téměř jediným proto, že by se za helper dala považovat i již dříve zmíněná metoda *Raw()* pro vypsání neoštřené proměnné. Avšak na internetových stránkách W3Schools je uveden pouze *ActionLink*, zbytek jsou formulářové helpery.

Demonstrován bude tedy jediný *ActionLink*. Tato demonstrace je znázorněna na obrázku 5.39.

```
@Html.ActionLink("Odkaz","stranka")
```

Obrázek 5.39: Ukázka užití helperu v ASP.NET MVC

ActionLink obvykle díky dvěma parametrům (text odkazu a název akce) vytvoří HTML značku *<a>* (tedy odkaz), která bude odkazovat na akci předanou v parametru pro ActionLink s textem odkazu, který byl taktéž předán. V konkrétním případě tedy *Odkaz*.

ASP.NET MVC sice strádá na helperech, ale vytvoření vlastního helperu je velice jednoduché. Pokud programátor používá Visual Studio, je to pár kliknutí, pár řádku kódu

a jednoduchý helper je na světě. Jediné, co je třeba udělat, je vytvořit si třídu s libovolným názvem (např. CustomHelpers) a v ní nadefinovat veřejnou statickou metodu, tedy vlastní helper. Není ovšem na škodu, aby programátor dodržoval jisté konvence a samotné helpery měl v jedné složce projektu, a ne všude možně. Vlastní helper se poté volá jako metoda dané třídy přes „zaviňáčovou" konvenci, jak je v šablonovacím systému Razor zvykem. Tvorba a volání vlastního helperu demonstruje ukázka na obrázcích 5.40 a 5.41.

```
namespace CMSDemo.Helpers
{
    public class CustomHelpers
    {
        public static string Pozdrav(string name)
        {
            return "Ahoj " + name;
        }
    }
}
```

Obrázek 5.40: Ukázka tvorby vlastního helperu v ASP.NET MVC

```
@{ var jmeno = "Pavle";}
@CustomHelpers.Pozdrav(jmeno)
```

Obrázek 5.41: Ukázka volání vlastního helperu v ASP.NET MVC

	Nette	Django	Rails	MVC
Počet helperů	3	2	1	4
Složitost užívání helperů	1	1	1	1
Složitost tvorby vlastního helperu	2	3	1	1
Celkově	2	2	1	2

Tabulka 5.5: Výsledky v kategorii "Rozšiřitelnost"

5.6 Práce s daty

5.6.1 Nette

V Nette se k datům přistupuje cestou View ->Presenter ->Model. Pomocí dependency injection se do proměnné presenteru vloží objekt modelu, jež obsahuje přístup k databázi. Poté v metodě *renderDefault()* předáme konkrétní šabloně data získaná z modelu. To by bylo stručně popsané získání dat z modelu, nyní se podíváme, jakým způsobem se data vypisují. Jelikož nám většinou presenter vrátí více výsledků z databáze, je dobré si tyto výsledky vypsat v cyklu. V Nette existují tři typy cyklů – for, foreach a while. Nejvhod-

nějším cyklem pro výpis většího počtu dat z databáze (a také nejvíce používaným) je cyklus foreach, díky výhodě možností použití iterátoru. Foreach vypadá následovně:

```
{foreach $data_z_db as $radek}
  {$radek->id}
{/foreach}
```

Prochází pole *$data_z_db*, každým průchodem se do proměnné *$d* nahrají data aktuálního záznamu. Poté se přes šipku přistupuje k jednotlivým atributům záznamu. Oproti jiným šablonovacím systémům Latte nesjednocuje způsob přístupu k atributům objektu a prvkům pole. U atributů objektu se používá šipková notace, v případě polí se používají pro přístup k prvkům hranaté závorky.

Ukázka cyklu se nachází na obrázku 5.2 v první kapitole o úvodních informacích o šablonovacím systému. Na obrázku je v cyklu foreach procházen výsledek vrácený z databáze a uložen do pole *$clanky*. Poté v každé iteraci je vypsáno nadpis a text jednoho článku (řádku) z databáze.

5.6.2 Django

Stejný princip přístupu k datům jako v Nette je i v Djangu (Template ->View ->Model). Ve view (z pohledu MVC lze View v Djangu chápat jako Controller) si pouze vytvoříme proměnnou, kterou naplníme buďto jediným záznamem pomocí metody *get()*, která má většinou parametr primárního klíče, nebo ji pomocí metody *all()* naplníme všemi daty z modelu. Existuje zde ještě metoda *filter()*, která vybere všechna data, která odpovídají zadanému požadavku. V případě metody *get()* se vrátí přímo objekt, na druhou stranu v případě užití metod *all()* nebo *filter()* se vrátí QuerySet, což je kolekce dotazů. Metody pro přístup k datům z modelu jsou vyjmenovány, ale jak říct view, z kterého modelu data chceme? Jednoduše napíšeme, že chceme objekt nebo objekty z konkrétního modelu a specifikujeme, jakou metodou je chceme získat, např. *Nazev.objects.all()*. Všechny vyjmenované metody slouží pro získání dat pomocí DB engine.

Ať už použijeme metodu *get* (objekt), nebo *all/filter* (QuerySet), vždy na jejich jednotlivé atributy přistupujeme pomocí tečkové konvence.

Pro hromadný výpis dat se používá cyklu for (resp. jediného cyklu, který lze použít v šablonách). Sice cyklus nese název for, ale oproti cyklu foreach použitého v Nette se moc neliší. Navíc má také pomocné proměnné (*first*, *last*, *counter*) a vypadá tedy následovně:

```
{% for radek in data_z_db %}
  {{radek.id}}
{% endfor %}
```

Princip cyklu for v Djangu je naprosto totožný s cyklem foreach v Nette. Prochází jednotlivé záznamy z vráceného QuerySetu a v každém průchodu si ukládá data aktuálního záznamu do proměnné *d*.

V ukázce na obrázku 5.4 byl použit právě cyklus for pro výpis všech článků. Prochází všechny záznamy z proměnné *clanky*, která obsahuje vrácený QuerySet z modelu. Poté se odkazuje na jednotlivé atributy řádků (id, nazev, text) pomocí tečky.

5.6.3 Ruby on Rails

Princip získání dat z modelu v Rails se ničím neliší od principů v předešlých frameworcích (View ->Controller ->Model). Samotné získání dat v controlleru je hodně podobné získání dat ve view u Djanga. Jediné, co je potřeba udělat, je vytvořit si proměnnou, do které uložíme vše, co nám model vrátí. Na jednotlivé proměnné z controlleru se odkazujeme jejich názvem. Na získání všech dat z modelu se používá metoda *all()*. Získání dat z modelu vypadá tedy následovně *NazevModelu.all()*. Pokud chceme získat specifická data, lze k tomuto účelu využít metodu *find()*. Tato metoda vyhledává podle id nebo metodu *where()*, kde napíšeme podmínku (např. nazev: 'ahoj'). Ve všech případech nám Model vrátí pole výsledků.

Pro přístup k jednotlivým atributům řádku z modelu se používá tečková konvence.

Jako cyklus pro výpis více výsledků mohou být použity cykly – for, while..do a each..do. Jako nejvíce užívaný byl i v testovací aplikaci použit cyklus each..do, tím se zde i budeme zabývat. Vypadá následovně:

```
<% @data_z_db.each do |radek| %>
  <%= radek.id%>
<%end%>
```

Cyklus *each..do* prochází všechny řádky, které byly vráceny z modelu, a ukládá je do proměnné *d*, díky které se poté odkazuje na jednotlivé atributy.

Na obrázku 5.6 lze vidět použití cyklu each..do, který v každém průchodu vypisuje požadované atributy (nadpis, kratky_text) z jednoho řádku výsledku uloženého v proměnné *articles*.

5.6.4 ASP.NET MVC

Abychom mohli v ASP.NET MVC přistupovat k datům, je nutné získat připojení k databázi v controlleru (podobně jako v Nette) a poté načíst data pomocí připojení z konkrétního modelu a použít metodu *ToList()*, aby se data uspořádala do kolekce a na závěr tyto data vrátit. Vypadá to tedy následovně: *databaze.Model.ToList()*. Tento princip nám vrátí všechna data z požadovaného modelu, abychom data mohli konkretizovat, je třeba užít jedné ze dvou metod *Find()* nebo *Where()*. Metoda *Find()* vyhledává pomocí id, kdežto metoda *Where()* vyhledává pomocí předaného parametru, který vypadá následovně *a=>a.Atribut.Contains("hledane_slovo")*. Co nám model vrátí, je v podstatě na nás. Můžeme použít metodu *ToList* pro vrácení do kolekce List, nebo *ToArray* pro vrácení do pole.

Pro přístup k jednotlivým atributům se používá tečková konvence, ať už jde o pole, nebo o kolekci List. Jako cyklus je pro výpis polí, a zvláště pak kolekcí, nejlepší cyklus foreach. Jeho syntaxe je velmi podobná syntaxi v Nette:

```
@foreach (var radek in data_z_db){
   radek.id
}
```

Princip je úplně totožný s principem v Nette. Prochází jednotlivé řádky vrácené z modelu a pomocí proměnné *d* poté přistupuje na jednotlivé atributy.

Na obrázku 5.8 vidíme použití cyklu foreach, který si do proměnné *c* ukládá aktuální řádek z modelu. Jak lze vidět, zde se přímo neprochází proměnná, která je naplněna daty (jak tomu bylo v jiných frameworcích), ale pokud je hledaný článek nalezen již při hledání dat v controlleru, vytvoří se instance modelu Article. Ta je pak předána do požadovaného view. Tudíž cyklus prochází právě tuto instanci modelu. Poté se v cyklu pomocí odkazování přes proměnnou *c* vypisují jednotlivé atributy záznamu (Nadpis, KratkyText). Další zajímavou věcí je v ukázce použití výrazu *@model*, kterým se specifikuje, že právě tento model view očekává.

Poznámka 5.1 Kapitola o práci s daty byla spíše kapitolou doplňkovou, mnoho věcí se zde nedá hodnotit. Pokud bych měl srovnat, jak všechny frameworky pracují s cyklem, a jak přistupují k datům, je to vesměs stejný postup. Všechny frameworky bych tedy ohodnotil známkou 1.

6 Zhodnocení

Na úvod bych uvedl souhrnnou tabulku všech frameworků a jimi získané známky v jednotlivých kategoriích.

	Nette	Django	Rails	MVC
Základní informace	1,5	1	1,5	2
Lokalizace	2	1,3	1,3	1
Testovatelnost	4	3	3	3,5
Bezpečnost	2,3	3,6	3	2,3
Rozšiřitelnost	2	2	1	2
Práce s daty	1	1	1	1
Výsledná známka	2,1	2	1,8	2

Tabulka 6.1: Celkové hodnocení frameworků

Jak tedy z tabulky vyplývá, jasným vítězem se stal framework Ruby on Rails – nasbíral nejnižší průměr známek. Framework Rails zvítězil celkově ve třech kategoriích. Stejně tak frameworky ASP.NET MVC a Django (oba druhé místo). Avšak tyto frameworky nedokázaly nasbírat stejný nebo nižší průměr známek jako Ruby on Rails. Nette zvítězilo ve dvou kategoriích a skončilo na třetím místě. Je však nutno dodat, že u všech frameworků je započítána kategorie „Práce s daty“, ve které si byly všechny frameworky rovnocenné.

Známky frameworků se ve výsledku však mnoho neliší a především není cílem vyhlásit jednoznačného vítěze. Proto bych nyní rád srovnání prošel po jednotlivých kategoriích a porovnal, kde si frameworky vedly dobře nebo naopak.

6.1 Základní informace

V této kategorii zvítězilo Django. Avšak nebyly zde velké rozdíly ve výsledcích a všechny frameworky obstály téměř na výbornou. S dědičností šablon nemá žádný framework problém a je snadno pochopitelná. Co frameworkům známkování kazilo, byla podpora v IDE. Django má jednu z nejlepší podpor v IDE díky pluginu PyDev. Nette s Rails se umístily stejně, editorů je mnoho, ale většina umí jen zvýrazňovat syntaxi, s našeptáváním mají problém. Na posledním místě zde skončilo ASP.NET MVC. Samotné Visual Studio je výborný editor, umí našeptávat, zvýrazňuje syntaxi. Ale je to jediný editor z mnoha, který tímto vyniká. Některé umí pouze zvýraznit syntaxi a jiné umí jak zvýraznit syntaxi, tak našeptávat, umí ale pracovat pouze s nižšími verzemi frameworku. Editorů pro ASP.NET MVC navíc mnoho není.

6.2 Lokalizace

Zde zvítězil framework ASP.NET MVC. Tento framework poskytl nejsnadnější nastavení lokalizace, tvorbu lokalizačních souborů, i lokalizace samotné. Na druhém místě skončily

současně frameworky Django a Rails. Django mělo nevýhodu v nastavování samotného spuštění lokalizace (mnoho konfigurace), na druhou stranu lokalizační soubory vytváří framework sám a je jednoduché se v nich vyznat. Rails má menší problém s tvorbou lokalizačních souborů, kdy je třeba dávat pozor na odsazení a mezery. Struktura je ve větším množství překladů docela nepřehledná. Avšak nastavení samotné lokalizace není opět nic složitého. Nette skončilo jako poslední z důvodu, že nemá (jako ostatní srovnávané frameworky) úplně přímou funkci překladače textů, a proto je třeba si tento překladač udělat, nebo použít plugin.

6.3 Testovatelnost

I u testovatelnosti lze vidět, že rozdíly jsou opravdu malé. Většina frameworků neposkytuje přímo nástroj pro testování šablon. Výjimkou je Django, které takový nástroj v jisté podobě poskytuje. Co se pluginů pro rozšíření frameworku o tuto funkcionalitu týče, ASP.NET a Ruby on Rails poskytují knihovny pro integraci testovacích nástrojů jako Selenium či Capybara. Vítězové jsou v této kategorii dva, Django a Ruby on Rails, a to velice oprávněně, protože Django jako jediný framework tvoří výjimku a poskytuje, byť jen sebemenší, testování šablon. Ruby on Rails má na druhou stranu mnoho pluginů pro integraci testovacích nástrojů.[22]

6.4 Bezpečnost

V kategorii „Bezpečnost“ jsou dva vítězné frameworky – ASP.NET MVC a Nette. Všechny testované frameworky umí automaticky escapovat, co však umí pouze dva vítězné frameworky, je kotextové escapování. Nette tuto funkcionalitu poskytuje automaticky a ASP.NET MVC pomocí knihovny AntiXSS. Dalším porovnávaným prvkem bylo i jiné řešení obrany proti XSS útoku, např. pomocí některé metody nebo knihovny/pluginu. Zde zabodovalo na třetím (druhém) místě umístěné Ruby on Rails a ASP.NET MVC, které poskytují takovouto metodu (sanitize). Rails však nepodporuje technologii kontextového escapování, a to mu ubírá na výsledku z bezpečnosti. Třeba se ho dočkáme v některé další verzi. Django zde dopadlo nejhůře a v průměru nasbíralo známku 3,6. To byl také nejhorší výsledek z celého testu. Nepodporuje ani kontextové escapování, ani nemá metodu pro alternativní obranu proti XSS.

6.5 Rozšiřitelnost

V rozšiřitelnosti vyhrálo Ruby on Rails, a to hlavně díky velkému množství helperů, které poskytuje. Také používání helperů a tvorba vlastních helperů je zde velice jednoduchá. Poté se stejně umístily zbylé tři srovnávané frameworky. Nette ztratilo hodnocení na počtu helperů a mírné složitosti tvorby vlastního helperu (oproti Rails). Django má větší počet helperů než Nette, avšak tvorba vlastního helperu zabere více času a konfigurace než u ostatních frameworků. ASP.NET ztratilo hodnocení pouze na základě malého počtu helperů, jinak užívání a tvorba helperů je zde velice jednoduchá.

6.6 Práce s daty

Jak jsem již zmínil poznámkou 5.1 na konci podkapitoly ve srovnání, všechny frameworky mají stejný princip přístupu k datům a výpis dat převážně provádí přes cykly foreach nebo for. Data jsou většinou vrácena v kolekci či poli. Proto zde frameworky obdržely známku 1.

Závěr

Cílem této bakalářské práce bylo porovnat čtyři různé frameworky v různých programovacích jazycích. Porovnání bylo zaměřeno tak, aby nebyl pouze stanoven vítězný framework, ale aby byly především odhaleny silné a slabé stránky frameworků a jejich šablonovacích systémů. Jako první byly stanoveny a popsány kritéria pro srovnání, následovalo porovnání všech těchto kritérií ve zvolených frameworcích a poté následovalo, kde se mi podařilo vystihnout, co kterému frameworku schází a v čem vyniká. Cílem bylo, mimo jiné, popsat pojmy framework a návrhový vzor, konkrétně Model-View-Controller, dále stručně popsat framework a programovací jazyk, ve kterém je framework napsán. I tento cíl se podařilo splnit v prvních třech kapitolách.

Z výsledků srovnání vyplývá, že vítězným frameworkem je Ruby on Rails. Můj názor na výsledek je, že bych na první místo umístil Ruby on Rails. Sice má své vady, co se týče bezpečnostních děr [25], ale to není jediný. Jinak mu v testování nedělalo problém téměř nic. Na druhé místo bych umístil Django, jeho výsledek v bezpečnosti nebyl tak dobrý a důvody jsou zřejmé (nemá žádnou alternativní metodu pro řešení XSS útoků, ani nepodporuje kontextové escapování). Stejné místo bych udělil ASP.NET MVC. Tomuto frameworku nedělalo problém téměř nic a první místo ve srovnání si zasloužil, avšak jeho popularita ve tvorbě internetových aplikací oproti Ruby on Rails není tak velká. Na poslední místo bych umístil Nette. Nette mě v některých ohledech zklamalo, na druhou stranu v jiných ohledech zase potěšilo. Nejvíce mě potěšilo u bezpečnosti, na druhou stranu lokalizace v tomto frameworku (bez pluginů) je oproti jiným frameworkům o dost náročnější, což mě celkem zklamalo, neboť framework by jen přeci měl šetřit čas.

Práce nebyla a není určena pro profesionální programátory, ale spíše pro začátečníky či mírně pokročilé, kteří snad ve srovnání svůj Framework naleznou.

Je však nutné podotknout, že pokud čtenář bude číst tuto práci rok po jejím napsání, vše může být naprosto jinak, neboť vývoj frameworků a všeobecně webových technologií postupuje velkým tempem kupředu.

Tato práce mi dala velice mnoho znalostí a zkušeností nejen ohledně frameworků a jejich šablonovacích systémů, ale i o samotné tvorbě aplikací v těchto užitečných „pomocnících“. Při tvorbě testovacích aplikacích jsem psal kód i ve dvou zbylých částech návrhového vzoru MVC, tedy Model a Controller. Již na začátku psaní této práce mě zajímalo, jak jak budou vybrané frameworky úspěšné a byl jsem velice překvapen výsledky, kterých jsem v této práci docílil.

Literatura

[1] Design patterns: elements of reusable object-oriented software. Boston: Addison-Wesley, 1995, s. 40. ISBN 0-201-63361-2.

[2] JFW. What is a Framework? [online]. 2002 [cit. 2013-11-16]. Dostupné z: http://www.jfwk.com/what_is.html

[3] Design patterns: elements of reusable object-oriented software. Boston: Addison-Wesley, 1995, s. 12. ISBN 0-201-63361-2.

[4] VAADIN.COM. Model-View-Presenter Pattern with Vaadin [online]. 2013 [cit. 2014-04-25]. Dostupné z: https://vaadin.com/web/magi/home/-/blogs/model-view-presenter-pattern-with-vaadin

[5] PINGDOM. A history of the dynamic web [online]. 2007 [cit. 2013-11-30]. Dostupné z: http://royal.pingdom.com/2007/12/07/a-history-of-the-dynamic-web/

[6] TIOBE. TIOBE Index for February 2014 [online]. 2014 [cit. 2014-02-08]. Dostupné z: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[7] GITHUB. Hhvm [online]. 2014 [cit. 2014-02-08]. Dostupné z: https://github.com/facebook/hhvm/wiki

[8] DEBIAN. Computer Language Benchmarks Game [online]. 2013 [cit. 2013-11-30]. Dostupné z: http://benchmarksgame.alioth.debian.org/u32/benchmark.php?test=all&lang=python3&lang2=php&data=u32

[9] MICROSOFT. Locked What Programming Language is Windows written in? [online]. 2009 [cit. 2013-11-30]. Dostupné z: http://social.microsoft.com/Forums/en-US/65a1fe05-9c1d-48bf-bd40-148e6b3da9f1/what-programming-language-is-windows-written-in?forum=windowshpcacademic

[10] GOOGLE. Google Trends [online]. 2004- [cit. 2013-12-14]. Dostupné z: http://www.google.cz/trends/explore#q=%2Fm%2F0505cl%2C%20Nette%2C%20%2Fm%2F06y_qx%2C%20ASP.NET%20MVC&cmpt=q

[11] W3SCHOOLS. AJAX Tutorial [online]. 1999- [cit. 2013-12-14]. Dostupné z: http://www.w3schools.com/Ajax/

[12] TECHNOPEDIA. Object-Relational Mapping (ORM) [online]. 2010- [cit. 2013-12-14]. Dostupné z: http://www.techopedia.com/definition/24200/object-relational-mapping–orm

[13] ZDROJÁK.CZ. Dependency Injection: motivace [online]. 2010- [cit. 2013-12-14]. Dostupné z: http://www.zdrojak.cz/clanky/dependency-injection-motivace/

[14] NETTE.ORG. Seznámení s Nette Frameworkem [online]. 2008- [cit. 2013-12-14]. Dostupné z: http://doc.nette.org/cs/2.1/getting-started

[15] ZDROJÁK.CZ. Django: Úvod a instalace [online]. 2009 [cit. 2013-12-14]. Dostupné z: http://www.zdrojak.cz/clanky/django-uvod-a-instalace/

[16] DJANGO ČESKÁ REPUBLIKA. Poznejte Django [online]. 2005- [cit. 2013-12-14]. Dostupné z: http://djangoproject.cz/

[17] DJANGO. FAQ: General [online]. 2005- [cit. 2013-12-14]. Dostupné z: https://docs.djangoproject.com/en/dev/faq/general/

[18] SLEPI. Design Principles in Ruby on Rails [online]. 2008 [cit. 2013-12-14]. Dostupné z: http://www.slepi.net/blog/programming/design-principles-in-ruby-on-rails.html

[19] BHW. ASP.Net MVC Principles – Part 1 [online]. 2013 [cit. 2013-12-14]. Dostupné z: http://www.thebhwgroup.com/blog/2013/10/asp-net-mvc-principles/

[20] BHW. ASP.Net MVC Principles – Part 2 [online]. 2013 [cit. 2013-12-14]. Dostupné z: http://www.thebhwgroup.com/blog/2013/10/asp-net-mvc-principles-part-2/

[21] THE RUBY TOOLBOX. Template Engines [online]. 2009- [cit. 2014-02-15]. Dostupné z: https://www.ruby-toolbox.com/categories/template_engines

[22] THE RUBY TOOLBOX. Browser testing [online]. 2009- [cit. 2014-03-14]. Dostupné z: https://www.ruby-toolbox.com/categories/browser_testing

[23] GOOGLE. Reducing XSS by way of Automatic Context-Aware Escaping in Template Systems [online]. 2009 [cit. 2014-03-22]. Dostupné z: http://googleonlinesecurity.blogspot.cz/2009/03/reducing-xss-by-way-of-automatic.html

[24] CODEPLEX. Microsoft Web Protection Library [online]. 2010 [cit. 2014-03-22]. Dostupné z: http://wpl.codeplex.com/

[25] LINUXEXPRESS. Vážná bezpečnostní chyba v Ruby on Rails [online]. 2013 [cit. 2014-04-19]. Dostupné z: http://www.linuxexpres.cz/novinky/vazna-bezpecnostni-chyba-v-ruby-on-rails

NETTE. Dokumentace [online]. 2008 - [cit. 2014-04-10]. Dostupné z: http://doc.nette.org/cs/2.1/

RAILSGUIDES. Ruby on Rails Guides [online]. 2004- [cit. 2014-04-10]. Dostupné z: http://guides.rubyonrails.org/

DJANGO. Django documentation [online]. 2005- [cit. 2014-04-10]. Dostupné z: https://docs.djangoproject.com/en/1.6/

MICROSOFT ASP.NET. Learn About ASP.NET MVC [online]. 2009- [cit. 2014-04-10]. Dostupné z: http://www.asp.net/mvc

RUBY, Sam, David THOMAS a David Heinemeier HANSSON. Ruby on Rails: průvodce agilním vývojem webových aplikací. Vyd. 1. Computer Press, 2011, 488 s. ISBN 978-80-251-3647-8.